НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

Кафедра системного програмування і спеціалізованих комп'ютерних систем

До захисту допущено

Завідувач кафедри
_____ Романкевич В.О.
(підпис) (ініціали, прізвище)
"___" червня 2020 р.

Дипломний проєкт

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Комп'ютерні системи та компоненти»

спеціальності **123 «Комп'ютерна інженерія»**

на тему: Автоматизація тестування вебдодатків_____
_____
_____

Виконав :

студент IV курсу, групи КВ-63
(шифр групи)

Воронін Микита Глібович _____ _____
(прізвище, ім'я, по батькові) (підпис)

Керівник асистент каф. СПіСКС Радченко К.О._____ _____
_____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант з нормоконтролю, доц.каф.СПСКС, к.т.н. Клятченко Я.М. _____
(назва розділу) (посада, вчене звання, науковий ступінь, прізвище, ініціали) (підпис)

Рецензент_____ _____
(посада, науковий ступінь, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2020 року

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

Кафедра системного програмування і спеціалізованих комп‘ютерних систем

Рівень вищої освіти – перший (бакалаврський)

Спеціальність 123 «Комп’ютерна інженерія»

Освітньо-професійна програма «Комп’ютерні системи та компоненти»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Романкевич В.О.
(підпис) (ініціали, прізвище)
«___» червня 2020 р.

ЗАВДАННЯ
на дипломний проєкт студента
Вороніна Микити Глібовича
(прізвище, ім’я, по батькові)

1. Тема проєкту Автоматизація тестування вебдодатків,
керівник проєкту асистент каф. СПіСКС Радченко К.О. _____ ,
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від «25» травня 2020р. №1181-С

2. Термін подання студентом проєкту _____

3. Вихідні дані до проєкту див. Технічне завдання

4. Зміст пояснювальної записки
- Аналіз видів вебдодатків та способів їх побудови
- Аналіз підходів до автоматизації тестування
- Порівняльна характеристика інструментів для програмної реалізації автоматизованого тестування вебдодатку
- Налаштування системи безперервної інтеграції до проєкту

5. Перелік графічного матеріалу (з зазначенням обов'язкових креслень, плакатів, презентацій тощо)

- Порядок виконання тестів у системі безперервної інтеграції. Схема алгоритму
- Взаємодія модулів системи. Схема структурна
- Життєвий цикл автоматизації тестування. Схема алгоритму
- Система безперервної інтеграції. Схема алгоритму

6. Консультанти розділів проєкту[1*]

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
нормоконтроль	к. т. н., доцент Клятченко Я.М.		

7. Дата видачі завдання _____

Календарний план

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів проєкту	Примітка
1	Вивчення літератури за тематикою проєкту	8.03.2020	
2	Розробка та узгодження технічного завдання	25.03.2020	
3	Аналіз існуючих рішень	12.04.2020	
4	Підготовка матеріалів розділів дипломного проєкту	17.04.2020	
5	Підготовка звіту до дипломного проєкту	12.05.2020	
6	Передзахист дипломного проєкту	20.05.2020	

Студент _____ _____
(підпис) (ініціали, прізвище)

Керівник проєкту _____ _____
(підпис) (ініціали, прізвище)

[1] * Консультантом не може бути зазначено керівника дипломного проекту.

АНОТАЦІЯ

Дипломна робота на тему «Автоматизація тестування вебдодатків» на здобуття освітньо-кваліфікаційного рівня «Бакалавр» зі спеціальності «Комп'ютерна інженерія», написана обсягом 63 сторінки і містить 15 ілюстрацій, 3 таблиці та 15 джерел з переліком посилань.

Метою даної роботи є побудова системи автоматизованого тестування програмного забезпечення вебдодатку за рахунок поєднання у собі кількох способів тестування та налаштування безперервної інтеграції для проєкту.

Методи досліджень. До уваги взяти існуючі на сьогоднішній день технології, написанні мовою Javascript, виконане їх порівняння та аналіз. Здійснюється реалізація системи з використанням безперервної інтеграції, управління цією конфігурацією.

В результаті дослідження наведено порівняльну характеристику та опис способів тестування на кожному рівні тестування вебдодатку, здійснено програмну реалізацію автоматизованого тестування прикладного програмного інтерфейсу, що покращить якість програмного забезпечення і полегшить його розробку. Результати досліджень можуть бути використані для побудови власного фреймворку автоматизованого тестування мовою Javascript та налаштування системи звітності до проєкту.

КЛЮЧОВІ СЛОВА: ВЕБДОДАТКИ, АВТОМАТИЗОВАНЕ ТЕСТУВАННЯ, БЕЗПЕРЕРВНА ІНТЕГРАЦІЯ, JAVASCRIPT, ПІРАМІДА ТЕСТУВАННЯ, ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ.

SUMMARY

Thesis on "Automation of testing web applications" for the educational qualification level "Bachelor" in "Computer Engineering" is written in 63 pages and contains 15 illustrations, 3 tables, and 15 sources with a list of references.

The purpose of this work is to build a system of automated testing of web application software by combining several methods of testing and to set up continuous integration for the project.

Research methods. Taking into account the current technologies, written in Javascript, performed their comparison and analysis. The system is implemented using continuous integration, management of this configuration.

The study provides a comparative description and description of testing methods at each level of web application testing, software implementation of automated testing of the application software interface, which uses continuous integration technology to work effectively with different methods of testing the software system. The research results can be used to build your own automated testing framework in Javascript and to configure the implementation of the process of continuous integration and reporting system.

KEY WORDS: WEB APPLICATIONS, AUTOMATED TESTING, CONTINUOUS INTEGRATION, JAVASCRIPT, TESTING PYRAMID, SOFTWARE.

Поз.	Формат	ПОЗНАЧЕННЯ	НАЙМЕНУВАННЯ	Кількість аркушів	№ прим.	Примітки
	А4	ІАЛЦ.045490.002 ТЗ	Автоматизація тестування	4		
			тестування			
			вебдодатків			
			Технічне завдання			
	А4	ІАЛЦ.045490.003 ТП	Автоматизація	2		
			тестування			
			вебдодатків			
			Відомість технічного			
			проекту			
	А4	ІАЛЦ.045490.004 ПЗ	Автоматизація	63		
			тестування			
			вебдодатків			
			Пояснювальна записка			
	А4	ІАЛЦ.045490.005 Д1	Порядок виконання	1		
			тестів у системі			
			безперервної інтеграції			
			Схема алгоритму			

Змін.	Арк.	№ докум.	Підпис	Дата	ІАЛЦ.045490.001 ОА		
Розробив		Воронін М.Г.			Автоматизація тестування вебдодатків **Опис альбому**	Літ.	Аркуш 1 / Аркушів 2
Перевірив		Радченко К.О.					
Консульт.						НТУУ «КПІ» ім. Ігоря Сікорського, ФПМ, КВ-63	
Н. контроль		Клятченко Я.М.					
Зав. каф.		Романкевич В.О					

Поз.	Формат	ПОЗНАЧЕННЯ	НАЙМЕНУВАННЯ	Кількість аркушів	№ прим.	Примітки
	А4	ІАЛЦ.045490.006 Д2	Взаємодія модулів	1		
			системи			
			Схема структурна			
	А4	ІАЛЦ.045490.007 Д3	Життєвий цикл	1		
			автоматизації тестування			
			Схема алгоритму			
	А4	ІАЛЦ.045490.008 Д4	Система безперервної	1		
			інтеграції			
			Схема алгоритму			
		Диск CD-ROM	Текст пояснювальної			
			записки.			
			Графічний матеріал			

Змін.	Арк.	№ докум.	Підпис	Дата	ІАЛЦ.045490.001 ОА	Арк. 2

ЗМІСТ

1. НАЙМЕНУВАННЯ ТА ГАЛУЗЬ РОЗРОБКИ 2

2. ПІДСТАВА ДЛЯ РОЗРОБКИ 2

3. ЦІЛЬ І ПРИЗНАЧЕННЯ РОБОТИ 2

4. ДЖЕРЕЛА РОЗРОБКИ 2

5. ТЕХНІЧНІ ВИМОГИ 2

5.1. Вимоги до програмного продукту, що розробляється 2

5.2. Вимоги до апаратного забезпечення 2

5.3. Вимоги до програмного та апаратного забезпечення користувача 3

6. ЕТАПИ РОЗРОБКИ 4

1 НАЙМЕНУВАННЯ ТА ГАЛУЗЬ РОЗРОБКИ

Назва розробки: «Автоматизація тестування вебдодатків».
Галузь застосування: розробка програмного забезпечення.

2 ПІДСТАВА ДЛЯ РОЗРОБКИ

Підставою для розробки є завдання для дипломного проектування на здобуття першого (бакалаврського) рівня вищої освіти, затверджене кафедрою системного програмування та спеціалізованих комп'ютерних систем Національного технічного університету України «Київський Політехнічний Інститут імені Ігоря Сікорського».

3 МЕТА І ПРИЗНАЧЕННЯ РОБОТИ

Основною метою даного проєкту є покращення якості ПЗ, шляхом побудови моделі автоматизованого тестування, що має забезпечити надійність роботи вебдодатку і простоту його подальшої розробки і підтримки.

4 ДЖЕРЕЛА РОЗРОБКИ

Джерелом інформації є різна науково-технічна та технічна література, технічна документація, публікації у інтернет виданнях та електронні ресурси у мережі Інтернет.

5 ТЕХНІЧНІ ВИМОГИ

5.1 Вимоги до програмного продукту, що розробляється

- кросбраузерність;
- використання сучасних технологій;
- система звітності;

5.2 Вимоги до апаратного забезпечення

- Процесор: Intel i5-7400 ;
- Оперативна пам'ять: 16 Гб;
- Монітор

5.3 Вимоги до програмного та апаратного забезпечення користувача

- Встановленний Node.js;
- Операційна система Windows, Linux;

6 ЕТАПИ РОЗРОБКИ

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів
1.	Вивчення літератури за тематикою проєкту	8.03.2020
2.	Розроблення та узгодження технічного завдання	25.03.2020
3.	Аналіз існуючих рішень	12.04.2020
4.	Підготовка матеріалів розділів до дипломного проєкту	17.04.2020
5.	Підготовка звіту до дипломного проєкту	12.05.2020
6.	Передзахист дипломного проєкту	20.05.2020

Поз.	Формат	ПОЗНАЧЕННЯ	НАЙМЕНУВАННЯ	Кількість аркушів	№ прим.	Примітки
	А4	ІАЛЦ.045490.004 ПЗ	Автоматизація	63		
			тестування			
			вебдодатків			
			Пояснювальна записка			
	А4	ІАЛЦ.045490.005 Д1	Порядок виконання	1		
			Тестів у системі			
			Безперервної інтеграції			
			Схема алгоритму			
	А4	ІАЛЦ.045490.006 Д2	Взаємодія модулів	1		
			системи			
			Схема структурна			
	А4	ІАЛЦ.045490.007 Д3	Життєвий цикл	1		
			автоматизації			
			тестування			
			Схема алгоритму			

Змін.	Арк.	№ докум.	Підпис	Дата	ІАЛЦ.045490.003 ТП		
Розробив		Воронін М.Г.			Автоматизація тестування вебдодатків **Відомість технічного проекту**	Літ.	Аркуш 1 / Аркушів 2
Перевірив		Радченко К.О.					
Консульт.						НТУУ «КПІ» ім. Ігоря Сікорського, ФПМ, КВ-63	
Н. контроль		Клятченко Я.М.					
Зав. каф.		Романкевич В.О.					

Поз.	Формат	ПОЗНАЧЕННЯ	НАЙМЕНУВАННЯ	Кількість аркушів	№ прим.	Примітки
	А4	ІАЛЦ.045490.008 Д4	Система безперервної	1		
			інтеграції			
			Схема алгоритму			
		Диск CD-ROM	Текст пояснювальної	1		
			записки.			
			Графічний матеріал			

Змін.	Арк.	№ докум.	Підпис	Дата	ДП.468300.003 ТП	Арк 2

Пояснювальна записка
до дипломного проєкту

на тему: Автоматизація тестування вебдодатків

Київ – 2020 року

ЗМІСТ

СПИСОК ТЕРМІНІВ, СКОРОЧЕНЬ ТА ПОЗНАЧЕНЬ .....

ВСТУП .....

МЕТА ДОСЛІДЖЕННЯ ТА ПОСТАНОВКА ЗАДАЧІ 5

1. АНАЛІЗ ВИДІВ ВЕБДОДАТКІВ ТА СПОСОБІВ ЇХ ПОБУДОВИ..... 6

1.1 МЕХАНІЗМИ СТВОРЕННЯ ТА ВИДИ ВЕБДОДАТКІВ 6

1.2 СТРУКТУРА ВЕБДОДАТКУ 12

ВИСНОВКИ ДО РОЗДІЛУ 1 15

2. АНАЛІЗ ПІДХОДІВ ДО АВТОМАТИЗАЦІЇ ТЕСТУВІАННЯ 16

2.1 ЖИТТЄВИЙ ЦИКЛ АВТОМАТИЗАЦІЇ ТЕСТУВАННЯ 16

2.2 ПІРАМІДА ТЕСТУВАННЯ У АВТОМАТИЗАЦІЇ 26

ВИСНОВКИ ДО РОЗДІЛУ 2 31

3. ПОРІВНЯЛЬНА ХАРАКТЕРИСТИКА ІНСТРУМЕНТІВ ДЛЯ ПРОГРАМНОЇ РЕАЛІЗАЦІЇ АВТОМАТИЗОВАНОГО ТЕСТУВАННЯ ВЕБДОДАТКУ 34

3.1 ВИБІР МОВИ ПРОГРАМУВАННЯ ДЛЯ АВТОМАТИЗАЦІЇ 34

3.2 ПОРІВНЯННЯ JS-ФРЕЙМВОРКІВ ДЛЯ UNIT-ТЕСТУВАННЯ..... 36

3.3 ПОРІВНЯННЯ JS-ФРЕЙМВОРКІВ ДЛЯ END-TO-END-ТЕСТУВАННЯ..... 40

ВИСНОВКИ ДО РОЗДІЛУ 3 48

4. НАЛАШТУВАННЯ СИСТЕМИ БЕЗПЕРЕРВНОЇ ІНТЕГРАЦІ ПРОЕКТУ..... 49

4.1 ПРОГРАМНА РЕАЛІЗАЦІЯ АВТОМАТИЗОВАНОГО ТЕСТУВАННЯ ПРИКЛАДНОГО ПРОГРАМНОГО ІНТЕРФЕЙСУ 47

4.2 ПІДКЛЮЧЕННЯ ТЕСТІВ ДО СИСТЕМИ БЕЗПЕРЕРВНОЇ ІНТЕГРАЦІЇ 54
4.3 ПРОГРАМНА РЕАЛІЗАЦІЯ АВТОМАТИЗОВАНОЇ ЗВІТНОСТІ ДО ПРОЄКТУ 58
ВИСНОВКИ ДО РОЗДІЛУ 4 60
ВИСНОВОК 61
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 62

АТ – автоматизоване тестування

ЖЦАТ – життєвий цикл автоматизації тестування

ПЗ – програмне забезпечення

VM – віртуальна машина

БД – база даних

ОС – операційна система

CI – сервер інтеграції

API – прикладний програмний інтерфейс

GUI – графічний інтерфейс користувача

ВСТУП

В сучасному світі програмне забезпечення має дуже широкий вплив майже на усі сфери нашої життєдіяльності. Складність розробки і підтримки кожного програмного продукту зростає з неймовірною швидкістю, тому й задача зменшення вартості та складності його розробки стає все більш актуальною. Майже усі сфери підприємництва, бізнесу, починаючи від індустрії розваг та послуг та закінчуючи медициною і сферою освіти починають впроваджувати нові програмні продукти, які повинні полегшити роботу своїм користувачам. Але не кожен замислюється над тим, наскільки важко налагодити процес розробки та тестування цих додатків і наскільки складна архітектура стоїть за тим інтерфейсом користувача, який ми бачимо.

Тим не менше, вимоги, які постають перед командами розробників, з кожним роком ростуть з великими темпами. В наші дні існують програмні рішення, над якими одночасно працюють сотні і тисячі співробітників одночасно. Вартість знайденої під час розробки помилки – не дуже велика, потрібно буде з'ясувати у програмному коді, що саме призвело до неї, виправити і ще раз протестувати. Але якщо кінцевий користувач знаходить якусь програмну помилку – вартість виправлення збільшується в рази, а інколи й навіть може коштувати життя, якщо ми кажемо про медицину або військову сферу. Саме тому, більшість сучасних ІТ-компаній витрачають великі бюджети на тестування свого ПЗ, коли здавалося можна було би обійтись без цього. Існує велика кількість методологій, напрямків та способів тестування.

Розділяють два основні види тестування – ручне та автоматизоване. Все більша кількість компаній бажає автоматизувати процес тестування у себе, але повністю відійти від ручного тестування все одно не вийде. Існують випадки, коли щось потрібно перевірити один раз і витрачати час на автоматизацію цього процесу просто немає сенсу.

МЕТА ДОСЛІДЖЕННЯ ТА ПОСТАНОВКА ЗАДАЧІ

Метою даного проєкту є покращення якості ПЗ, шляхом побудови моделі автоматизованого тестування, що має забезпечити надійність роботи вебдодатку і простоту його подальшої розробки і підтримки.

Науково-практична задача, що розв’язується в даній дипломній роботі, включає наступні завдання:

1) Аналіз та порівняння існуючих технологій у автоматизації тестування вебдодатків;
2) Побудова моделі автоматизації на всіх рівнях тестування;
3) Написання автоматизованих тестів користувацького інтерфейсу
4) Налагодження процесу безперервної інтеграції даної моделі
5) Підключення системи звітності до проєкту

1. АНАЛІЗ ВИДІВ ВЕБДОДАТКІВ ТА СПОСОБІВ ЇХ ПОБУДОВИ

1.1. Механізми створення та види вебдодатків

Вебдодаток – це наперед створене програмне забезпечення, яке на звернення від користувача, віддає наперед створену гіпертекстову інформацію. Можна вважати що сайт і вебдодаток це – тотожні поняття. Архітектурно, майте кожен вебдодаток створено за системою клієнт-сервер. Користувач відправляє запит вебдодатку, вебдодаток його обробляє та відправляє через мережу до веб-сервера. У відповідь сервер формує веб-сторінку і відправляє її назад вебдодатку також мережею. Вебдодаток у свою чергу вже відображає отриману від сервера інформацію користувачу. Як правило, вебдодаток використовує браузер, як інтерфейс користувача.

Сучасний вебдодаток є унікальною розробкою, орієнтованою на вирішення певного кола завдань. Він починає бути затребуваним, завдяки тому, що дозволяє ефективніше вести бізнес-процеси. Для збільшення кількості послуг та підвищення їх якості й простоти уявлення, компанії замовляють розроблення вебдодатків. Гроші, вкладені у його розроблення - швидко окупаються.

Виділяють основні види вебдодатків:

- форуми
- форми відправки повідомлень
- веб-пошта
- системи голосування
- рахівники
- гостьові книги
- форми для завантаження та скачування
- движки сайту

- веб-системи, які надають комплекс послуг для своїх користувачів[1].

Більшість систем у сучасному світі не ідеальні, тому слід згадати й вразливості, які вони мають:

- під час роботи з файлами, коли він передається програмі ззовні (GET або POST);
- некоректна обробка для вхідних даних;
- некоректне зберігання, передача та обробка паролів;
- неврахування особливостей роботи програм для завантаження файлів на сервер;
- некоректна логіка під час роботи веб-програми, що при окремих допустимих вхідних даних призводить до непередбачуваних наслідків;
- вивід інформації про помилки програми або доступу до бази даних, в той час коли виводиться додаткова службова інформація, яка може бути використана злочинцями;
- не достатньо захищена робота з базами даних (під час роботи з паролями, виведенням службової інформації, при обробці величезної кількість запитів до бази);
- вразливості обробки вхідних даних при роботі з базою даних (різні види SQL-ін’єкцій);
- неоптимізований програмний код, який може призводити до значних навантажень на веб-сервер (у випадку передачі некоректних вхідних даних);
- вразливість вебдодатків та систем до DoS та DDoS атак;
- відсутня валідація даних під час вводу від користувача [1].

Класифікуємо різні типи вебдодатків. Ця класифікація заснована на тому, як вебдодатки показують прийнятий ними вміст. Виходячи з цього, ми маємо до 6 різних типів вебдодатків.

Статичні веб-програми:

Якщо ви вирішили створити статичний вебдодаток, перше, що потрібно знати, це те, що цей тип вебдодатків відображає дуже малий вміст і не відрізняється особливою гнучкістю.

Статичні веб-програми зазвичай розробляються в HTML та CSS, але це не єдині платформи для розвитку статичного додатка; ви можете використовувати jQuery та Ajax відповідно до власних зручностей. Ви також можете легко відображати анімовані об'єкти, такі як банери, GIF, відео та інші.

На жаль, змінити вміст статичних вебдодатків непросто. Для цього спочатку потрібно завантажити HTML-код, потім змінити його і, нарешті, відправити назад на сервер. Ці зміни можуть бути внесені тільки веб-майстром або командою розробки, яка запланувала та розробила веб-сайт.

Прикладами розроблених статичних вебдодатків - є професійні порфоліо або цифрові резюме. Їх достатньо легко спроектувати та розробити. Так само сторінка, що представляє компанію, може також використовувати цей тип вебдодатків для відображення їх контактної інформації.

Динамічні вебдодатки:

Динамічні вебдодатки набагато складніші на технічному рівні. Вони використовують бази даних для завантаження даних, а їх вміст оновлюється кожного разу, коли користувач звертається до них або у випадку коли на сторінці трапляється будь-яка подія. Список подій, можливих в DOM, дуже довгий: drag (перетягування), click (клік мишею), touch (торкання), load (завантаження), input (введення), error (помилка), change (зміна), resize (зміна розміру) і т. д. Події можуть спрацьовувати для будь-якої частини

документа, внаслідок взаємодії з ним користувача або браузера. Вони не просто починаються і закінчуються в одному місці - вони циркулюють по всьому документу, проходячи свій власний життєвий цикл. Це і робить події DOM настільки гнучкими і корисними. Розробники і тестувальники повинні розуміти як працюють події DOM, щоб мати можливість використати їх потенціал і побудувати інтерактивний інтерфейс.[2] Зазвичай динамічні вебдодатки мають панель адміністрування (називається CMS), де адміністратори можуть виправляти або змінювати вміст контенту, незалежно текст це чи зображення.

Для розробки динамічних вебдодатків можна використовувати різні мови програмування, яких існує велика кількість. Для написання Back-end частини, найчастіше використовують:

- Сценарні: Python, Ruby, PHP, Node.js, TypeScript
- Функціональні мови: Elixir, Scala, Clojure, Haskell, Erlang
- Мультипарадигмні мови: Goland, Rust.
- Корпоративні рішення: Java, .NET[3].

Front-end - це розробка користувальницького інтерфейсу та функцій, які працюють на клієнтській стороні вебдодатку. Це все, що бачить користувач, відкриваючи веб-сторінку і те з чим він взаємодіє.

Основними компонентами frontend-розробки є:

HTML (HyperText Markup Language) - мова для розмітки документів, створення структури сторінки: заголовки, абзаци, списки і тд.

CSS (Cascading Style Sheets) - мова для опису і стилізації зовнішнього вигляду документа. Завдяки CSS-коду браузер розуміє, яким чином відображати елементи. CSS задає кольори і параметри шрифтів, визначає, як будуть розташовуватися різні блоки сайту. Він дозволяє виводити один і

той же документ у різних стилях, виведення передачі на екран або читання голосом.

JavaScript - це мова, що створювалась з ціллю оживити веб-сторінки. Його завдання - реагувати на дії користувача, обробляти кліки мишкою, переміщення курсора, натискання клавіш. Він посилає запити на сервер і завантажує дані без перезавантаження сторінки, дозволяє вводити повідомлення та багато іншого[4].

У цьому типі додатків оновлення вмісту дуже просте, і сервер навіть не потребує доступу до змін, які слід вносити. Крім того, це дозволяє реалізувати велику кількість функцій, таких як форуми або бази даних. Веб додаток може бути змінений відповідно до налаштувань адміністратора.

Shop Online або E-Commerce:

Якщо вебдодаток є інтернет-магазином, його розробка, ймовірно, буде схожа на розробку веб-сайту m-commerce або e-commerce. Цей тип розробки додатків складніший, оскільки він повинен допускати електронні платежі, які можна здійснювати з кредитних карток, PayPal чи інших платіжних систем. Розробник також повинен створити панель управління для адміністратора, яка буде використовуватися для переліку нових продуктів, оновлення їх, видалення записів та керування програмами та платежами.

Веб-програма повинна підходити для мобільних пристроїв так само, як і мобільний додаток, що дозволить взаємодіяти з нею, так як це був би нативний додаток.

Портальний вебдодаток

Ми маємо на увазі тип додатку, який здійснює доступ до різних розділів або категорій через домашню сторінку. Ці програми можуть

включати багато речей: форуми, чати, електронну пошту, браузери, області, до яких можна отримати реєстрацію, останній вміст тощо.

Анімовані веб-програми:

Анімація неминуче пов'язана з технологією Flash. Цей підхід у програмуванні дозволяє відображати вміст з анімованими ефектами. Цей тип додатків виглядає більш сучасно та є однією з ключових технологій, яку використовують дизайнери та креативні директори. Недоліком, притаманним розробці анімаційних вебдодатків, є те, що цей тип технології не підходить для цілей розміщення в Інтернеті та оптимізації SEO, оскільки пошукові системи не можуть правильно читати інформацію, яку вони містять.

Вебдодатки із системою управління вмістом:

Вміст у таких вебдодатках слід постійно оновлювати. Коли мова йде про розробку такого типу вебдодатків, потрібно реалізувати можливість управління вмістом (CMS), аби адміністратор міг використовувати цю CMS, щоб робити зміни та оновлювати контент самостійно.

Ці менеджери вмісту інтуїтивно зрозумілі та дуже прості в обробці. Деякі приклади систем управління контентом:

WordPress: безсумнівно, найпоширеніший навколо управління вмістом. Один із лідерів ринку.

Joomla: Ця CMS поступається лише WordPress. Він не має таку велику кількість користувачів, але має сильну спільноту, а також дуже інтуїтивно зрозумілий.

Drupal: це безкоштовне програмне забезпечення CSM - дуже адаптивне, особливо рекомендується для побудови спільнот[5].

1.2 Структура вебдодатку

В більшості випадків для створення ПЗ використовується мінімум 3 основних компоненти:

1. Клієнтська частина – це фактично та система, якою ми користуємося у себе на комп'ютері. В її ролі може виступати браузер. Сам браузер вміє лише правильно відображати інформацію, отриману з будь якого ресурсу.

2. Серверна частина – програма, яка знаходиться на сервері та обробляє запити від користувача (через браузер). Вона взаємодіє з клієнтом, щоб дати змогу отримати інформацію. Відповідає за збереження та обробку даних, підтримку веб-інтерфейсу користувача, взаємодію клієнта з обліковою системою. Для прикладу, на малюнку серверна частина реалізована на мові PHP, але на сьогоднішній день існує велика кількість мов програмування для побудови Back-End. В залежності від дій користувача, це може бути як перехід по посиланню, так і клік по відповідній кнопці – відправляється запит на сервер, сервер обробляє цей запит і виконує певну роботу, в залежності від того, що описав програміст. Нова сторінка може бути відправлена користувачу у вигляді HTML файлу, яку вже повинен обробити браузер та відобразити для користувача.

3. База даних – являє собою програмне забезпечення на сервері, задачею якого є збереження і швидка видача даних після запиту користувача. Сама база даних розташовується безпосередньо на сервері. БД відповідає за зв'язок із сервером. Серверна частина вебдодатку звертається до бази даних, робить запит до даних, які потрібні для формування сторінки, запит на яку, отримано від користувача. Інтегровані (вбудовані) СУБД використовуються як складові інших програмних продуктів, наприклад словників, електронних енциклопедій, інтернет магазинів тощо. Для цих

систем не потрібно щось інсталювати окремо, вони можуть мати обмежений набір функцій для управління БД, наприклад зміна даних має бути неможливою. Доступ до даних повинен здійснюватися шляхом використання засобів прикладної програми, до якої повинна бути інтегрована СУБД.

На Рисунку 1.1 зображена схема основних класифікацій СУБД.

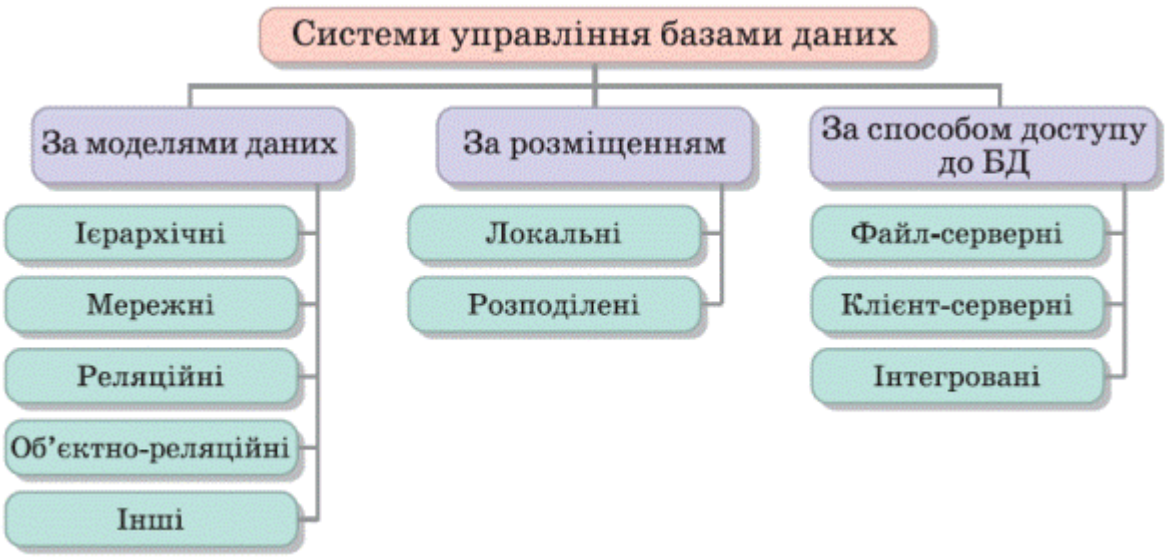


Рисунок 1.1 – Схема основних класифікацій систем для управління базами даних

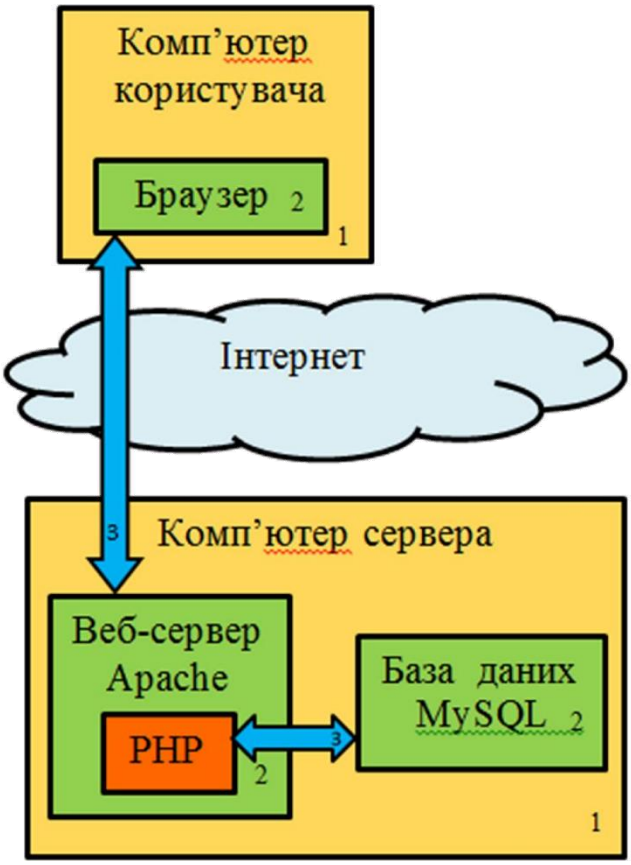


Рисунок 1.2 – Графічна схема взаємодії між базою даних, клієнтом та сервером: 1 – зв'язок між комп'ютерами, 2 – програми, 3 – зв'язок між програмами.

На рисунку 1.2 зображена схема взаємодії складових вебдодатку. Браузер, через мережу Інтернет відправляє HTTP-запити до веб-серверу. Веб-сервер у відповідь викликає скрипт, написаний розробником вебдодатку. Цей скрипт звертається до БД, якщо в цьому є необхідність. В результаті він повертає клієнту веб-сторінку, яку й повинен відобразити браузер.

Висновки до розділу 1:

Підсумовуючи, під час дослідження, були розглянуті технології створення та види вебдодатків. Надійність вебдодатків є необхідністю, про яку не потрібно забувати і приділяти достатню кількість уваги. Були розглянуті основні методи тестування ПЗ, для задоволення цієї необхідності. В ході аналізу сформувалася повна картина структури побудови автоматизації тестування системи та можлива програмна реалізація.

2. АНАЛІЗ ПІДХОДІВ ДО АВТОМАТИЗАЦІЇ ТЕСТУВІАННЯ

2.1 Життєвий цикл автоматизації тестування

Рисунок 2.1 – Схема життєвого циклу автоматизації тестування

Керівники проектів та розробники досить часто стикаються з проблемою створення додатків в умовах недостатньої кількості ресурсів та постійно скорочуваного графіка. Незалежно від того, наскільки досвідчені програмісти працюють над створенням певного ПЗ, все одно потрібно перевіряти чи дійсно воно відповідає вимогам потрібно. Таким чином, організації переходять до автоматизації тестування для ефективного досягнення цієї мети.

Коли ми говоримо про автоматизацію тестування, багато хто з нас вважає, що це лише частина SDLC (життєвого циклу розробки програмного забезпечення), але для досягнення найкращих результатів, потрібно дотримуватися саме повного циклу, відомого під назвою - життєвий цикл автоматизації тестування.

Реалізація життєвого циклу автоматизації тестування виконується паралельно з процесом життєвого циклу розробки програмного забезпечення.

Сам життєвий цикл є багатоетапним процесом, який підтримує діяльність, необхідну для використання та впровадження автоматизованого інструменту тестування, розробки та запуску тест кейсів, розробки тестового дизайну, побудови та обробки даних тесту та середовища.

У методології тестова конструкція побудована для відображення тестових зусиль та для надання проекту й тестовій групі розуміння щодо обсягу робіт з тестування.

Існує 6 основних етапів методології тестування життєвого циклу автоматизації:

- Визначення об'єму робіт з автоматизації тестування
- Вибір правильної технології для автоматизації
- Тест план + Тест Дизайн + Тест Стратегія
- Налаштування тестового середовища
- Розробка коду для запуску автоматизованих тестів і їх запуск
- Аналіз і генерація тестових звітів[6]

Визначення об'єму робіт з автоматизації тестування:

Це перший етап життєвого циклу АТ, і він спрямований на визначення наскільки взагалі доцільно застосовувати автоматизацію. Тут потрібно врахувати скільки часу є на побудову процесів та наскільки важко створити модель фреймворку автоматизації. Крім того, важливо провести аналіз

техніко-економічного обґрунтування на ручному пакеті тестових кейсів, який дозволяє інженерам-автоматизаторам розробляти програмні сценарії.

У цьому конкретному етапі слід обережно ставитися до наступних речей:

1. Які модулі додатку можна автоматизувати, а які ні?
2. Які тест кейси можна автоматизувати та як їх автоматизувати?
3. Такі фактори, як вартість, чисельність команди та досвід також слід враховувати.

Перед початком автоматизації тестів слід виконати наступні перевірки техніко-економічного обґрунтування:

- Можливість автоматизації тестових випадків
- Автоматизація автоматичної автоматизованості

Згадаймо всі компоненти програмного інтерфейсу та виділимо найбільш важливі для користувача. В першу чергу слід звернути увагу на ті, несправність яких може призвести до втрати найбільшої кількості грошей. Також необхідно визначити відсоток компонентів користувальницького інтерфейсу, які потрібно автоматизувати за допомогою інструменту автоматичного тестування. Спробуймо знайти засоби тестування автоматизації, які допоможуть автоматизувати компоненти інтерфейсу з невеликими змінами. Це полегшить роботу на наступному етапі.

Вибір правильної технології для автоматизації:

АТ дуже залежить від інструментів. Ось чому пошук правильного інструменту тестування автоматизації є критичною фазою для ЖЦАТ. Коли ми шукаємо інструмент автоматизації, нам потрібно пам’ятати про бюджет, технології що використовуються в проекті, знання розробників та

тестувальників всередині команди, гнучкість тощо. Потрібно обирати інструмент, який підтримується розробниками і має добре оформлену документацію.

Наприклад, якщо ми шукаєте автоматизований інструмент для перевірки сумісності браузера, нам потрібно чітко розуміти різноманітність підтримуємих браузерів. Можливість зйомки відео-звітів, метаданих скриптів автоматизації серед різних браузерів та пристроїв. Механізм виділення та відстеження помилок, спосіб логування програмного коду та інші.

Тест план + Тест Дизайн + Тест Стратегія:

Це найважливіша фаза методології ЖЦАТ, яка визначає, як досягти основної мети АТ. Планування тестової автоматизації - це перше і головне, на що потрібно звернути увагу на цьому етапі.

Вибір інструменту залежить від технологій, які використовуються в додатку. Потрібно повністю розуміти свій продукт перед початком автоматизації.

Наприклад, якщо це настільний додаток - знайдіть на якій мові він побудований. Для вебдодатку, знайте про можливості тестового фреймворку, його особливості роботи з компонентами, які ви використовуєте.

На етапі планування тестування, команда тестування приймає рішення про стандарти та рекомендації щодо створення тестової процедури, обладнання, програмного забезпечення та мережі для підтримки тестового середовища, попередній графік випробувань, вимоги до тестових даних. Повинна бути чітка процедура відстеження дефектів та пов'язаний з цим

інструмент відстеження контролю конфігурації тесту та середовищ інсценування.

Команда тестових інженерів розробляє тестову архітектуру для опису структури тестової програми та способів управління процедурами тестування після розроблення моделі тестової стратегії.

Після проектування створюється тестова архітектура, де описана структура всієї програми тестування, а також шляхи управління цією тестовою процедурою.

Не забудьте врахувати наступні речі при плануванні тестової стратегії:

Зберіть всі ручні тестові кейси з інструменту управління тестом, щоб визначити, який тестовий випадок потрібно автоматизувати.

Визначте, які рамки слід використовувати після розуміння плюсів і мінусів інструменту тестування. Побудуйте тестовий набір для тестового випадку автоматизації в інструменті для управління тестами.

У плані тестування обов’язково зазначайте передумови, ризики та залежність між інструментом та програмою. Шукайте схвалення щодо стратегії тестування у клієнтів або зацікавлених сторін.

Налаштування тестового середовища:

Як видно з назви, цей етап життєвого циклу автоматизації тестування передбачає налаштування машини або віддаленої машини, де будуть виконуватися тестові кейси. Навіщо нам потрібні віддалені машини? Тому що, якщо ми не живемо в ідеальному світі, наші користувачі використовуватимуть різні машини для доступу до нашого веб-сайту чи вебдодатку в Інтернеті.

Відслідковувати коректність роботи вебдодатку на різних пристроях - одне, але нам також слід бути обережними щодо різних браузерів та версій браузера. Оскільки наш веб-сайт може відображатися по-різному від одного браузера до іншого. Тестування сумісності веб-переглядачів, також відоме як крос-браузерне тестування, - це процедура, коли ми перевіряємо веб-сайт або вебдодаток у кількох версіях браузера, щоб переконатися, що ми створили безперебійну роботу для усіх наших користувачів.

Етап налаштування навколишнього середовища потребує ретельного планування, ми повинні переконатися, що зможемо максимально охопити тестове покриття для якомога більшої кількості сценаріїв. Потрібно правильно налаштувати навколишнє середовище, встановити програмне забезпечення, мережеві ресурси та обладнання, вдосконалити бази даних тестів та розробити сценарії тестового покриття з неперервною інтеграцією у процес розробки.

Що стосується крос-браузерного тестування браузера, налаштування сотень браузерів та версій браузера на численних пристроях може бути дуже складним завданням, а придбання лабораторії пристроїв - не доступний варіант для всіх. Тут почали грати хмарні інструменти, такі як – Amazon Web Services, Azure Devops, Google Web Services які пропонують понад 2000+ браузерів та версій браузера та розміщують VM для численних настільних та мобільних пристроїв.

Основні сфери для налаштування тестового середовища:

- Тестові данні, – не потрібно постійно створювати велику кількість тестових даних для перевірки функціональності. Якщо є можливість – потрібно заздалегідь створити все необхідне для

прогону тестових випадків. Це може заощадити велику кількість часу на виконання тестів і зберегти від невірних результатів.

- Середовище для тестування навантаження, – важливо мати окреме середовище для навантаження й аналізу можливостей обробки веб-трафіку, підтримки великої кількості користувачів і обробки достатньої кількості запитів одночасно.
- Контрольний список усіх систем, модулів та додатків, які підлягають тестуванню.
- Ізольований сервер баз даних
- Тестування на різних клієнтських операційних системах.
- Тестування на максимальній кількості браузерів з різними версіями браузера.

Переконайтесь, що ви протестуєте свій веб-сайт у низькій та високій мережі, щоб зрозуміти різницю між часом надання та загальним виглядом веб-сайту чи вебдодатка.

Документація є ключовою. Переконайтеся, що ви охоплюєте всі посібники по конфігурації / посібники з установки / посібники користувача в центральному репозиторії.

Налаштування тестового середовища включає такі завдання:

1. Ліцензії на інструменти
2. Інструменти налаштування, такі як сучасні текстові редактори та засоби їх порівняння.
3. Впровадження фреймворку автоматизації
4. Доступ AUT і дійсні облікові дані
5. Ліцензії на надбудови

Більшість організацій має окреме тестове середовище для тестування програмного забезпечення. Найкращий підхід - скопіювати дані з користувацького середовища і використовувати їх для тестування. Це

допоможе інженеру-випробувачу розкрити проблеми, не псуючи виробничі дані.

Кращими практиками для налаштування управлінням тестовим довкіллям, можна зазначити:

- Знання тестового середовища та навчіть членів групи тестування.
- Необхідне програмне забезпечення, ліцензії та обладнання.
- Контрольний список засобів автоматизації та їх конфігурації.
- Крос-браузерне тестування, щоб забезпечити покриття тестів на численних браузерах та версіях щодо пріоритету та частки ринку.
- Використання трафіку в режимі реального часу, щоб переконатися, що ваші зміни є більш стійкими.
- Планування використання тестового середовища.

Розробка коду для запуску автоматизованих тестів і їх запуск:

Після встановлення тестового середовища настає час виконати тестовий скрипт. Отже, ця фаза життєвого циклу автоматизації тестування присвячена виконанню всіх тестових сценаріїв.

Для виконання сценаріїв, підписані та перевірені тестові кейси доставляються команді тестування автоматизації.

Важливо переконатися що всі тестові сценарії працюють правильно. Отже, перед розробкою тестового сценарію потрібно подбати про наступні речі:

- Створення тестових сценаріїв повинно бути на основі фактичних вимог.

- Розробити загальні методи функцій, які можна використовувати протягом усього процесу тестування.
- Створіть багаторазовий, структурований та простий сценарій, щоб будь яка людина могла це зрозуміти.
- Проведіть перегляд коду іншою людиною з команди, для кращої якості продукту і покращення розуміння всього фреймворку членами команди.
- Використовуйте звітність для більш чіткого зображення результатів і загального прогресу у АТ.

Після успішної розробки тестових сценаріїв слід пам'ятати про наступні речі:

1. Тестовий сценарій повинен включати всі функціональні аспекти відповідно до тестового випадку.
2. Запускати тестові сценарії в різних середовищах та на різних платформах.
3. Якщо можливо, паралельне виконання може бути здійснено для економії часу та зусиль.
4. Якщо збій стався через певну функціональність, напишіть звіт про помилку.

Для виконання тестових сценаріїв, команда тестування повинна відповідати графіку, прийнятому для виконання процедури. На цьому етапі готується оцінка результатів тестів та документація про результати випробувань.

Плани, розроблені для тестування модулів, системи, прийняття користувача та інтеграції, виконуються для тестування системи в цілому. Профілювання коду проводиться під час тестування одиниць. Профілювання виявляє випадки, коли нераціональне масштабування алгоритмів, використання ресурсів та інстанцій.

Аналіз і генерація тестових звітів:

Після того, як всі типи тестувань виконались, команда тестування проводить аналіз для виявлення конкретної функціональності або компонентів, в яких виявлені певні проблеми.

Результати, отримані під час аналізу, можуть підтвердити, чи можуть виконані тестові сценарії / процедури виявити помилки.

Це остання фаза життєвого циклу тестування з автоматизації, і на цьому етапі створюються звіти про випробування. Ось чому тестові звіти мають вирішальне значення для аналізу того, наскільки добре ваш вебдодаток реагує на негаразди. Ви можете використовувати старий аркуш програми Excel, однак, сучасні тестові генератори надають звіти про додаток та всі тестові випадки, виконані за допомогою сценарію автоматизації на хмарній сітці Selenium Grid.

2.2 Піраміда тестування у автоматизації

Рисунок 2.2 – Піраміда ідеального тестування

На рисунку 2.2 зображена так звана ідеальна піраміда автоматизації тестування.

Почати слід з того, що взагалі робить тести автоматичними. Який саме тест можна назвати автоматичним? Насамперед той, виконання якого не потребує присутності людини. Способів для запуску та налагодження існує велика кількість. Є можливість налаштувати запуск тестів після окремих дій програміста, таких як новий коміт чи просто за певним розкладом.

На нижньому рівні знаходяться автоматичні модульні тести, ціллю яких являється перевірити що окремі кусочки коду, частіше за все функції, коректно працюють окремо одна від одної. Головна ідея в тому, щоб

подальші зміни вже існуючого коду не призвели до поломки вже протестованої частини програми. Як правило, написанням модульних тестів займається сам розробник коду або колега з команди. Час витрачений на один юніт-тест в рази менше ніж на будь який рівнем вище. Таким чином, юніт-тестування - це перший рівень для боротьби з багами. При розробці великих додатків модульні тести відіграють важливу роль. Якщо над продуктом починає працювати більше ніж одна людина, стає дуже важко вносити зміни у вже існуючий код, не зламавши жодного компоненту системи. На допомогу й приходять автоматичні юніт-тести, які можуть бігти у системі неперервної інтеграції після будь якої зміни у коді.

Але й слід звернути увагу на те, що у випадку коли ви самостійно розроблюєте невеликий додаток, написання модульних тестів може тільки сповільнити і поскладнити сам процес розробки.

Основна різниця між компонентним і модульним тестуванням полягає у тому, що компонентне тестування у якості параметрів функцій використовує об'єкти і драйвери, а у модульному – конкретні значення. [7]

В інтеграційних тестах перевіряються програмні модулі, які інтегровані між собою локально і перевіряються як група. Типове ПЗ складається з окремих модулів, створених різними програмістами. Основною ціллю цього рівня є знаходження дефектів під час взаємодії цих частин. Потрібно впевнитись, що навіть у випадку коли кожен модуль ідеально працює окремо, нічого не пошкоджується при їх взаємодії. Можна перевіряти як взаємодію компонентів так і різних систем. Для відокремлення окремих частин системи використовують так звані заглушки, які можуть емулювати або просто припиняти роботу окремих частин вебдодатку. Розділяють основні 3 підходи інтеграційного тестування:

Знизу вгору (Bottom Up Integration)

Всі низькорівневі процедури, модулі та функції збираються разом і тільки потім тестуються. Після цього береться наступний рівень модулів з метою проведення інтеграційного тестування. Даний підхід буде корисним, якщо практично всі модулі, які повинні бути створені – вже готові. За результатами тестування, такий підхід, може допомогти поступово знаходити вразливості на кожному рівні додатку.

Зверху вниз (Top Down Integration)

Спочатку тестуються усі високорівневі модулі та поступово один за одним додаються низькорівневі. Всі модулі низького рівня симулюються за допомогою заглушок з аналогічною функціональністю, потім по мірі готовності вони замінюються реальними активними компонентами. Таким чином відбувається тестування зверху вниз.

Великий вибух ("Big Bang" Integration)

Всі модулі збираються разом у вигляді закінченої системи і вже потім проводиться інтеграційне тестування. Такий підхід дуже добре підходить для збереження часу. Але, якщо тест кейси та їх результати будуть записані не вірно, то сам процес інтеграції може сильно ускладнитись, що стане перешкодою при досягненні основної мети інтеграційного тестування командою тестування.[8]

Прикладний програмний інтерфейс (API – англ. Application Programming Interface) надає можливість здійснювати обмін і зв'язок даними у двох програмних системах. Система ПЗ, що реалізовує API, повинна містити функції, які можуть бути виконані за допомогою іншої системи ПЗ. Щоб програми могли спілкуватися між собою, їх API повинен бути побудований по єдиному принципу. Самий популярний на сьогоднішній

день - REST. Це стандарт архітектури взаємодії додатків, який використовує HTTP. Тестування API виконують опираючись на бізнес-логіку програмного продукту. Для його тестування використовують спеціальні інструменти, які дозволяють робити запити і перевіряти достовірність вихідних даних.

Чим вище ми просуваємося по піраміді тестування, тим більших навичок і знань потрібно для реалізації тестування. Тестування API потребує розуміння системи і того як в ній влаштовані запити. Помилки HTTP сервера розподіляють на діапазони:

- 100-199 Інформаційні. Повідомляють що запит прийнятий і обробляється.
- 200-299 Запит оброблений успішно, сервер відправив клієнтові потрібний документ.
- 300-399 Запит змінений і агентові потрібно почати впроваджувати дії, з метою задоволення зміненого запиту.
- 400-499 Проблеми на стороні клієнту.
- 500-599 Серверні помилки

Різниця між модульним тестуванням і тестуванням API наведена у таблиці 2.1

Таблиця 2.1 – Порівняльна таблиця API і модульного тестування

Модульне тестування	Тестування API
Зазвичай тести пишуть розробники	Тести пишуть тестувальники, менше розробники
Тестуються окремі методи, функції	Тестується функціональність системи
Розробник може звернутися до програмного коду	Тестувальник, як правило, не може звернутися до програмного коду
Включене тестування з урахуванням графічного інтерфейсу	Тестуються лише функції з API
Тестування лише для базових функцій	Тестуються всі функціональні частини
Застосування обмежене	Широка сфера застосування
Зазвичай проходить під час написання програмного коду	Проходить після створення основного функціоналу

Довгий час SOAP був основним протоколом для обміну даними між веб-службами, але оскільки в наші дні розробникам все більше необхідно створювати компактні та швидкі додатки, більш гнучка REST архітектура швидко завоювала популярність. Розглянемо основні відмінності, двох архітектур, які зображені у таблиці 2.2. та спробуємо визначити основні переваги та недоліки. [8]

Таблиця 2.2 – Порівняльна таблиця REST і SOAP архітектур

	SOAP	REST
Значення	Simple Object Access Protocol	Representational State Transfer
Дизайн	Стандартизований протокол з чіткими заданими правилами оформлення	Архітектурний стил, який просто має свої рекомендації до оформлення
Підхід	Function-driven	Data-driven
Безпечність	WS-Security з підтримкою SSL. Вбудована ACID compliance.	Підтримує HTTPS and SSL.
Формат повідомлень	Тільки XML.	Plain text, HTML, XML, JSON, YAML та інші
Трансферні протоколи	HTTP, SMTP, UDP, and others.	Тільки HTTP
Затратність ресурсів	Велика	Меньша

Обидві архітектури дають можливість створювати власний API. SOAP, ймовірно, продовжить застосовуватися підприємствами, які потребують великого рівня безпеки та мають велику кількість складних транзакцій. Найгарнішими прикладами є банківські установи, системи управління ідентифікацією, платіжні шлюзи, компанії які надають телекомунікаційні послуги. Навіть одна з найбільших платіжних систем – PayPal, повністю побудована на API-інтерфейсах SOAP. Слід також не

забувати і про підтримку вже існуючих систем. У відомих вебдодатків вже може існувати велика кількість користувачів, які все ще підключаються до своїх сервісів через SOAP API. Наприклад, деякі вебдодатки можуть мати частину своєї веб-сервісної архітектори на SOAP, іншу - на REST.

В свою чергу у 2018 році REST став найпопулярнішим вибором розробників для побудови загальнодоступного API. Існує велика кількість соціальних мереж, які мають відкритий REST API-інтерфейс, що дає можливість розробникам дуже легко інтегрувати свої вебдодатки з будь якою платформою. Ці публічні API мають добре описану документацію, звідки можна взяти всю необхідну інформацію.

Підсумовуючи, основними перевагами SOAP є великий рівень безпеки, стандартизованість та розширюваність. З мінусів можна виділити – великий час на виконання, складність і менша гнучкість.

REST, в свою чергу, має кращі можливості для розширення, швидше виконується, більш гнучкий та краще працює з сучасними браузерами. З мінусів можна відзначити – менший рівень безпеки та погану роботу з розподіленими середовищами.

На верхньому рівні піраміди знаходяться UI End-to-End тести. З їх допомогою можна повністю перевірити систему чи сам продукт і бути впевненим, що весь інтерфейс користувача працює коректно. Вони займають найбільшу частину часу і можуть одночасно перевіряти велику кількість компонентів. В нас є можливість перевірити що кожен елемент на сторінці відображається відповідно до вимог, відбуваються переходи між сторінками, розмітка не пливе. Такий тест повністю емульє дії звичайного користувача і може одночасно перевіряти роботу декількох сервісів. Існує велика кількість інструментів, написаних різними мовами програмування, які надають можливість налагодити процес автоматичної перевірки

вебдодатків. Вони запускаються на останньому етапі і можуть гарантувати найбільше покриття і найкращу якість для кінцевого продукту.

E2E тестування – це складна інженерна робота, проведення мостів з датчиками і проведення всіх необхідних перевірок і ситуацій – чи хоча б опис цих сценаріїв.

Висновки до розділу 2:

Отже, рухаючись уверх по піраміді тестування, слід пам’ятати що з кожним новим рівнем, такі параметри як важкість, складність ті ціна зростають, а швидкість їх виконання зменшується. Тому, в ідеалі, кількість написаних тестів повинна збільшуватись згори вниз. Так і в тестуванні, звичайно, E2E тестування потрібно, тільки тоді, коли у нас відпрацювали як модульні тести, так і тести вже більш високого рівня. Буде на раціонально будувати всю систему тестування тільки за рахунок End-to-End тестів, але це більше питання виділених бюджетів і стратегій, які застосовуються при створенні ПЗ, бо саме End-to-End тести можуть гарантувати найбільшу надійність, бо саме вони емулюють дії реальних користувачів. [9]

3. ПОРІВНЯЛЬНА ХАРАКТЕРИСТИКА ІНСТРУМЕНТІВ ДЛЯ ПРОГРАМНОЇ РЕАЛІЗАЦІЇ АВТОМАТИЗОВАНОГО ТЕСТУВАННЯ ВЕБДОДАТКУ

3.1 Вибір мови програмування для автоматизації

Приступаючи до автоматизації, кожна команда приймає рішення керуючись потребами замовника або на основі своїх навичок та знань. Зараз з'являється все більше можливостей обирати мову автоматизації виходячи з стека технологій конкретного проекту. Як правило, намагаються обрати мову програмування, яка використовується на Back-End чи Front-End. Це дає можливість залучати більшу кількість членів команди у процес автоматизації, що покращує якість написання коду і надає можливість всім приймати участь у процесах написання та перевірки коду.

Різниця між мовами все більше зменшується і стає непомітною. Тому, переглядаючи сучасні тренди в автоматизації, можна побачити, що все більша кількість проектів використовують JavaScript для різних рівнів і видів тестування. Як приклад можна привести статистику Github за 2019 рік за популярністю мов програмування, рисунок 3.1.[10]. Оскільки ми знаємо що Front-End завжди пишеться за допомогою Javascript / TypeScript, а все більша кількість проектів обирає Node.js як мову для Back-End, у нас є можливість побудувати весь проект лише використовуючи Javascript. Тому для нас вибір на користь Javascript має бути найбільш оптимальним.

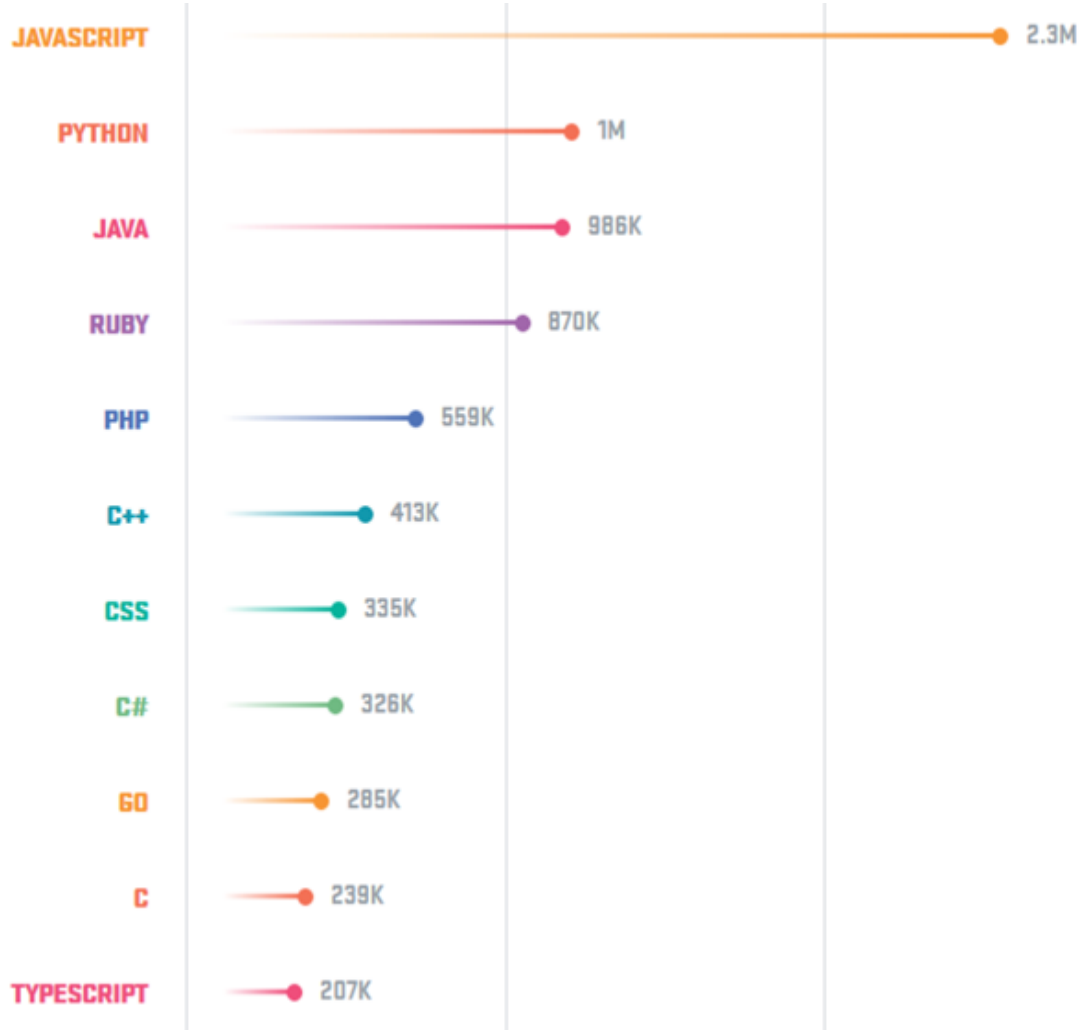


Рисунок 3.1 – Популярність мов програмування у 2019 році.

Як ми бачимо JavaScript більше ніж в 2 рази попереджає своїх найближчих конкурентів Java та Pyhton, але й слід звернути увагу на вже існуючі рішення на ринку автоматизації. Це теж відіграє важливу роль при виборі відповідної мови.

3.2 Порівняння JS-фреймворків для Unit-тестування

Рисунок 3.2 – Найпопулярніші фреймворки для Unit-тестування, написанні на Javascript / TypeScript

Jest

Ймовірно, один з найпопулярніших фреймворків тестування javascript, який має понад 22 000 зірок на github, був побудований та постійно підтримується командою з Facebook. Не потребує налаштування конфігурацій, рекомендований від React, і найпростіший у використанні. Jest має дуже вражаючий рівень прийняття у 2019 році спільнотою javascript.

Він має велику швидкість роботи та зручний інтерфейс користувача. Також передбачає тестування знімків і постачається із вбудованим інструментом покриття коду. Це неймовірно швидко і один з найкращих варіантів для початківців, які хочуть пройти тестування свого коду JavaScript. В Інтернеті також є багато ресурсів та документації [11].

Основні переваги JEST:

- Сумісний з NodeJS, React, Angular, VueJS та іншими проектами на основі Babel

- Стандартний синтаксис із підтримкою документації

- Дуже швидко та високоефективно
- Управління тестами з більшими об'єктами можливо за допомогою Live Snapshots

Mocha

MochaJS - це найпопулярніший тестовий фреймворк, який підтримує тестування бекенда та фронтенда. MochaJS - це гнучка база для розробки тестів у міру необхідності. Він запускає тести асинхронно на двигуні Chrome v8 або будь-якому іншому браузері.

Пропонує розробникам лише базову структуру тестування. Функціональність для тверджень, шпигунів, макетів потім додається за допомогою інших бібліотек / плагінів.

Якщо ви хочете гнучку конфігурацію, включаючи бібліотеки, які вам особливо потрібні, то додаткове налаштування та конфігурація, необхідні для Mocha, - це те, що ви обов'язково потрібно перевірити

На жаль, вищезазначений момент має і зворотний бік, який повинен включати додаткові бібліотеки для перевірок. Це означає, що налаштувати трохи складніше, якщо не довше, ніж інші.

Mocha включає структуру тесту як глобальну, що дозволяє економити ваш час.

Гнучкість у твердженнях та заглушках дуже корисна. В цілому Mocha охоплює основи і дозволяє розробникам розширювати їх.

До основних переваг Mocha можна віднести:

- Працює як для фронтенду, так і для бекенда
- Підтримка відладчика NodeJS

- Забезпечує чисту базу для розробки тестів відповідно до зручності розробника
- Підтримується будь-який веб-переглядач, включаючи бібліотеку для Google Chrome
- Підтримує заглушку з об’єктів для виконання запуску гнучких тестів

Jasmine

Jasmine - надає вам все, що вам потрібно з коробки. Це імітатор поведінки користувачів, який дозволяє виконувати тестові випадки, подібні до поведінки користувачів на вашому веб-сайті. Jasmine корисний для тестового інтернету на предмет видимості, чіткості натискань, а також чуйності інтерфейсу користувача в різних режимах дозволу. Jasmine дозволяє автоматизувати поведінку користувачів з мінімальними затримками та зачекат, щоб імітувати фактичну поведінку користувача.

Основні переваги використання Jasmine:

- Нижні накладні витрати через майже нульові зовнішні залежності
- Поставляється майже з кожним необхідним інструментом поза коробкою
- Підтримує тести написані для фронтенду, а також тести бекенду
- Написання коду досить схоже на написання натуральною мовою
- Обширна документація по використанню з іншими фреймворками
- Велика кількість ознайомчих онлайн курсів

AVA

Це мінімалістичний фреймворк тестування легкої ваги, яка використовує асинхронний характер Javascript. Він в основному зосереджений на запущених тестах для коду на основі NodeJS. AVA може виконувати тести паралельно.

Це дозволяє вам майже повний контроль над тим, що ви робите. Він в основному зосереджений на запущених тестах для коду на основі NodeJS. Деякі з переваг включають:

- Тести виконуються асинхронно та одночасно
- Швидше, ніж більшість інших тестових рамок
- Простіший синтаксис тестів Javascript
- Очисник стеку відстежує будь-які виявлені потенційні помилки

Karma

Це продуктивне середовище для тестування, яке підтримує всі популярні фреймворки опису тестів всередині себе. Він надає вашій програмі підтримку виконання тестів у різних середовищах. Він має широку підтримку виконання тестів на різних пристроях та додатках.

Основний чинник вибору Karma полягає в його підтримці інтеграції з двигунами CI / CD та наступними особливостями:

- Можна використовувати для запуску тестів на браузерах, безголових середовищах, таких як PhantomJS, а також на пристроях
- Підтримує тести, написані в більшості популярних фреймворків
- Дозволяє дистанційно запускати тести на інших пристроях, просто надсилаючи файли

- Підтримує налагодження тестових випадків за допомогою Chrome, а також WebStorm

3.3 Порівняння JS-фреймворків для End-To-End-тестування

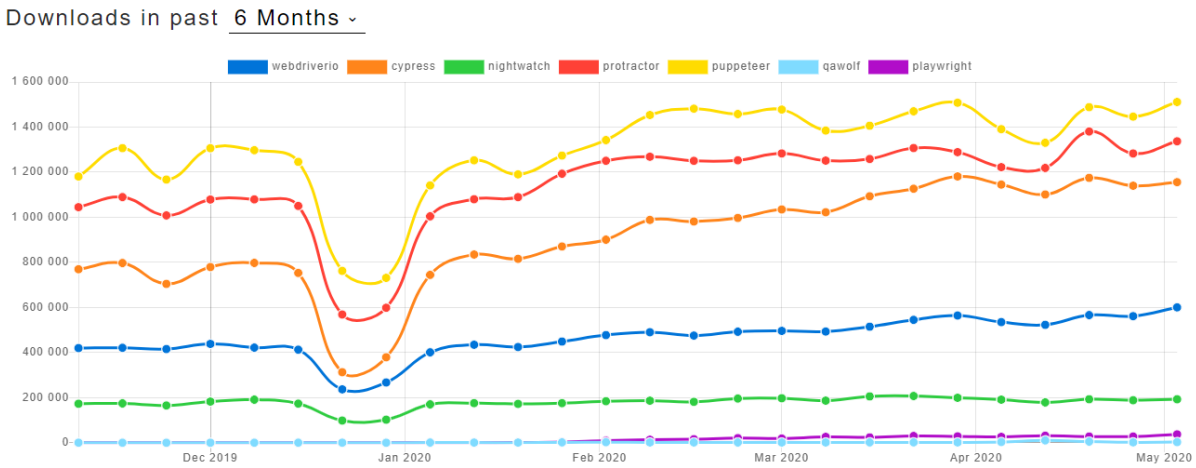
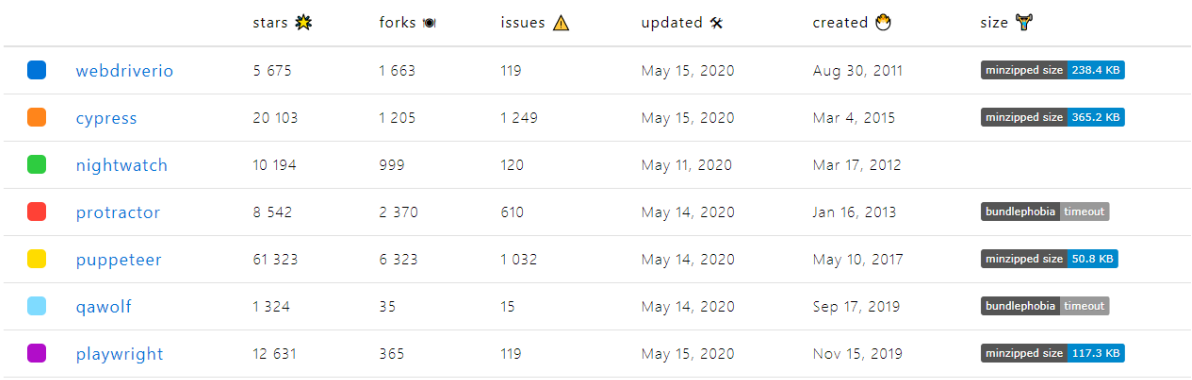


Рисунок 3.3 – Статистика скачувань фреймворків для End-To-End-тестування, написаних на Javascript / TypeScript за грудень 2019 – травень 2020 років

При виборі фреймворка для тестування, слід звернути увагу на його популярність на GitHub, підтримку від товариства, кількість відкритих проблем і сумісність з технологіями, які використовуються у вас в системах, а також на особливостях кожної програми, використовуваної для автоматизації. [13]



	stars	forks	issues	updated	created	size
webdriverio	5 675	1 663	119	May 15, 2020	Aug 30, 2011	minzipped size 238.4 KB
cypress	20 103	1 205	1 249	May 15, 2020	Mar 4, 2015	minzipped size 365.2 KB
nightwatch	10 194	999	120	May 11, 2020	Mar 17, 2012	
protractor	8 542	2 370	610	May 14, 2020	Jan 16, 2013	bundlephobia timeout
puppeteer	61 323	6 323	1 032	May 14, 2020	May 10, 2017	minzipped size 50.8 KB
qawolf	1 324	35	15	May 14, 2020	Sep 17, 2019	bundlephobia timeout
playwright	12 631	365	119	May 15, 2020	Nov 15, 2019	minzipped size 117.3 KB

Рисунок 3.4 – Інформація щодо актуальності і підтримки фреймворків для End-To-End-тестування, написаних на Javascript / TypeScript

Фреймворк	priority	**webdriverio**	**cypress**	**testcafe**	**puppeteer**	**protractor**
Screenshots	high	+	+	+	+	+
Parallel Testing	high	+	+	+	+/-	+
Easy debug	high	+/-	+/-	+/-	+/-	+/-
Sending native keyboard/mouse events	high	+	-	+	+	+
Fast tests execution	high	+/-	+		+	+
CI: good compatibility	high	+	+	+	+	+
Synchronous code execution	medium	+	+/-	+/-	+/-	+/-
Cross-browser testing	medium	+	+	+	+/-	+
Iframes support	medium	+	+/-	+	+	+
Mobile Testing	medium	+	+/-	+	+/-	+
Easy base-install for test execution	medium	+/-	+	+	+	+
Detailed documentation	medium	+	+	+		+

Multiple domains	medium	+	-	+	+	+
Files: uploading/downloading	medium	+	+/-	+	+	+
Low entry threshold	medium	+	+	+		
Access to browser console	low	+	+	+	+	+
Shadow DOM	low	+	+/-	+	+	+
Time Travel	low	-	+		-	-
Automatic Waiting	low	+/-	+	+	-	+/-
Network Traffic Control	low	+/-	+		+	-
Videos	low	+	+	+	-	+/-
Visual Testing	low	+	+	+/-	+/-	+/-
Custom asserts	low	+	+	+/-	-	
Retries: scenario or step	low	+	+/-		-	+/-
Multiple tabs	low	+	-	-	+	+
Multiple windows	low	+	-	+/-	-	+
User-friendly command syntax	low	+	+	+/-		+
TypeScript	low	+	+	+		+
Possibillity to intercept requests		+	+	+	+	-
Possibility to mock responses		+	+	+	+	-
Bundled with Chromium		-	+		+	
Reporters: good compatibility		+	+	+		+/-

Таблиця 3.1 – Порівняльна таблиця фреймворків для End-To-End-тестування, написаних на Javascript / TypeScript

У таблиці статуси: +: підтримується; +/-: частична підтримка або якась специфіка; -: не підтримується.

Protractor. Добре підходить для проектів, написаних на Angular, для яких є встроєні функції для конкретних завантажувачів Angular-елементів

Плюси:

- Protractor - це єдиний фреймворк, який із коробки підтримує кастомні визначення AngularJS-елементів. Якщо у вас Angular, використовуйте Protractor.
- Зручна підтримка TypeScript та різних фреймворків для тестування одиниць (Jasmine, Mocha, Cucumber та інші).
- Велика кількість допоміжних функцій

Мінуси:

- Нема підтримки мобільного тестування.
- Написаний як обертка над WebDriverJS. Якщо у вас будуть проблеми з WebDriverJS, вони автоматично будуть в ProtractorJS.
- Зупинилася підтримка від розробників

Базовий приклад тесту на ProtractorJS:

```
// spec.js
describe('Protractor Simple test', function() {
  it('should enter two numbers', function() {
    browser.get('https://www.madewithangular.com/');
    element(by.model('one')).sendKeys(1);
    element(by.model('two')).sendKeys(2);

    element(by.id('enter')).click();

    expect(element(by.binding('latest')).getText()).
        toEqual('10'); // That is wrong!
```

```
  });
});
```

Nightwatch.js

Плюси:

- Схожий з WebdriverIO, і він є кастомною імплементацією W3C WebDriver API.
- Легко додати нову функцію.
- Не потрібно вибирати між Jasmine і Mocha.

Мінуси:

- Нема підтримки Mocha та мобільного тестування.
- Менше підтримують, ніж у WebdriverIO та Cypress.

Базовий приклад тесту на Nightwatch.js:

```
module.exports = {
  after: function(browser) {
    console.log(Browser loaded.')
  },

  'Nightwatch simple test': function (browser) {
    browser
      .url('https://nightwatch.netlify.com/')
      .waitForElementVisible('[data-nw=name-input]')
      .setValue('[data-nw=name-input]', 'Text you want to enter')
      .pause(1000)
      .assert.containsText('[data-nw=welcome-message]',  'Welcome  your
text')
      .end()
```

}

Cypress

Плюси:

- На відмінність від більшості E2E - фреймворків, не використовує Selenium. Архітектурна роботи з браузером написана повністю та Cypress не використовує віддалених команд через мережу, а працює напряму з програмою.
- Швидке і зручне налаштування. Необхідно залишити лише одну команду для встановлення всіх пакетів, і немає необхідності ставити все окремо.
- Будь-яка документація в одному місці, не потрібно окремо шукати, як правильно робити перевірку або як використовувати Chai.
- Необхідний інтерфейс використовується з описом, які команди виконуються, що краще показує, що відбувається на сторінках у відповідний момент.

Мінуси:

- Плюс, який в той же час є і мінусом, - це той факт, що Cypress не використовує Selenium для end-to-end-тестування. Що породжує багато проблем в роботі. Робота з браузером не завжди стабільна, і на сьогоднішній день на GitHub є відкритими більш ніж 900 проблем.
- Немає підтримки мобільного тестування, і, судячи з коментарів творців нативной підтримки, ніколи і не буде.
- Підтримка обмеженої кількості браузерів: Canary, Chrome, Chromium, Electron.
- Платна паралелізація тестів. Хочете запускати в кілька потоків - доведеться платити.

```
describe(Cypress  simple Test', () => {
  it('clicking "type" navigates to a new url', () => {
    cy.visit('https://example.cypress.io')

    cy.contains('button').click()
    // should be redirected to a new url
    cy.url().should('include', '/commands/actions')
  })
})
```

WebdriverIO

Плюси:

- WebdriverIO - це кастомна імплементація W3C WebDriver API. Не потрібно прив'язуватися до імплементації WebDriverJS.
- Підтримка синхронного коду. Можна забути про асинхронний JavaScript.
- Зручне базове налаштування за допомогою вбудованого інтерфейсу для командного рядка wdio.
- Підтримує майже всі тестові фреймворки BDD і TDD.
- Підтримує зручну бібліотеку 'webdrivercss' для порівняння CSS-стилів елементів на сторінці.

Мінуси:

- Велика відмінність між останніми версіями (WebdriverIO 4 і 5).
- Гірше для автоматизації AngularJS, ніж Protractor.

```
describe('Simple WebdriverIO Test', () => {
  beforeEach(async(() => {
```

```
  TestBed.configureTestingModule({
   declarations: [
    AppComponent
   ],
  }).compileComponents();
 }));

 it('should add the app', () => {
  const myFixture = TestBed.createComponent(AppComponent);
  const application = myFixture.componentInstance;
  expect(application).toBeTruthy();
 });
```

У висновку, перелічимо переваги і недоліки використання JavaScript-фреймворків для автоматизації тестування, які ми відзначили вище.

Переваги:

- Швидкість написання тестів значно вище, ніж на Java або C #.
- Нижче поріг входу для старту проекту.
- Більше взаємодії всередині, якщо члени команди володіють однією мовою програмування.
- Велика кількість готових рішень дуже різних проблем, які виникають.

Недоліки:

- Менш стабільні рішення. Іноді може вийти так, що проблема є на стороні фреймворку.
- Для написання стабільних тестів потрібно розуміння, як працює JavaScript.

- У кожного рішення / мови / технології є свої переваги і свої недоліки. З вибором технологічного стека визначатися вам. Цей вибір залежить від складу вашої команди, від знань і досвіду її учасників, а також від завдань, які ви хочете вирішити. Якщо умови дозволяють, обов'язково дивіться в бік технологій, які активно розвиваються.

Висновки до розділу 3:

Отже, під час аналізу були розглянуті найпопулярніші на сьогоднішній день інструменти для автоматизації тестування вебдодатків. Досліджена їх продуктивність, заміряна швидкодія та характеристики, та побудовані графіки залежностей характеристик. Аналізуючи графіки, були зроблені висновки про сильні і слабкі сторони методів та зроблений висновок про їх ефективність. Були розглянуто перспективи і можливості застосування технологій безперервної інтеграції. Реалізовані приклади побудови програм для кожної технології. Були виділені конкретні інструменти і бібліотеки, які необхідні для реалізації програмної системи автоматизації тестування вебдодатків.

4. НАЛАШТУВАННЯ СИСТЕМИ БЕЗПЕРЕРВНОЇ ІНТЕГРАЦІЇ ПРОЕКТУ

4.1 Програмна реалізація автоматизованого тестування прикладного програмного інтерфейсу

Систему тестування інтерфейсу користувача будемо реалізовувати для сторінки http://scs.fpm.kpi.ua/

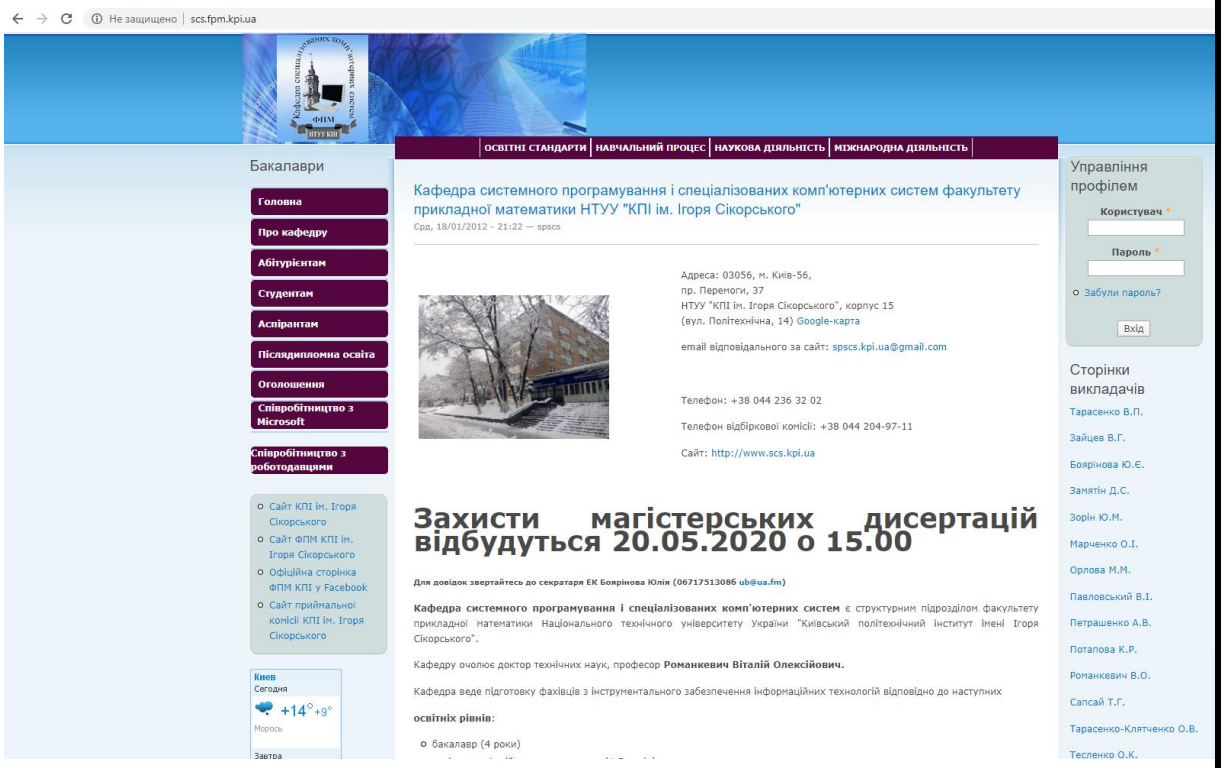


Рисунок 4.1 – Головна сторінка http://scs.fpm.kpi.ua/

Для цього нам буде потрібно встановити Node.js - середовище, яке може виконувати JavaScript-код. QA Wolf - це бібліотека Node.js, для якої потрібно запустити Node.js. npm поставляється в комплекті з Node.js і означає Node Package Manager. Це допомагає керувати пакетами, які потрібно запустити.

Щоб підтвердити, що у вас встановлені Node.js та npm, виконайте такі команди в CLI:

вузол -v

npm -v

Тепер, встановимо Node.js та додамо QA Wolf до нашого проекту [14].

Спочатку виберемо проект, де буде доданий QA Wolf. Потрібно створити новий проект або вибрати вже існуючий. Щоб створити новий проект, виконаємо наступне в командному рядку (необов'язково змінюючи назву проекту):

mkdir my-awesome-project

cd my-awesome-project

npm init -y

Щоб налаштувати QA Wolf потрібно створити файл qawolf.config.js у корені вашого проекту, а також встановити необхідні залежності.

npm init qawolf

Вам буде запропоновано вказати:

1. Каталог, де будуть створені тести (rootDir). За замовчуванням: .qawolf
2. Ваш постачальник послуг CI, тому при необхідності може бути створений файл робочого процесу для запуску ваших тестів у CI. За замовчуванням: GitHub Actions

QA Wolf також виявить, якщо ви використовуєте TypeScript, і оновить його конфігурацію, щоб створити тести в TypeScript замість JavaScript, якщо це застосовується.

Після закінчення встановлення залежностей виконайте наступне, щоб переконатися, що QA Wolf був встановлений успішно:

npx qawolf

Далі створимо простий тестовий сценарій, який буде перевіряти, що на сторінці для бакалаврів є “ПОСІБНИК з виконання бакалаврських дипломних проєктів (бакалаврських дипломних робіт)”

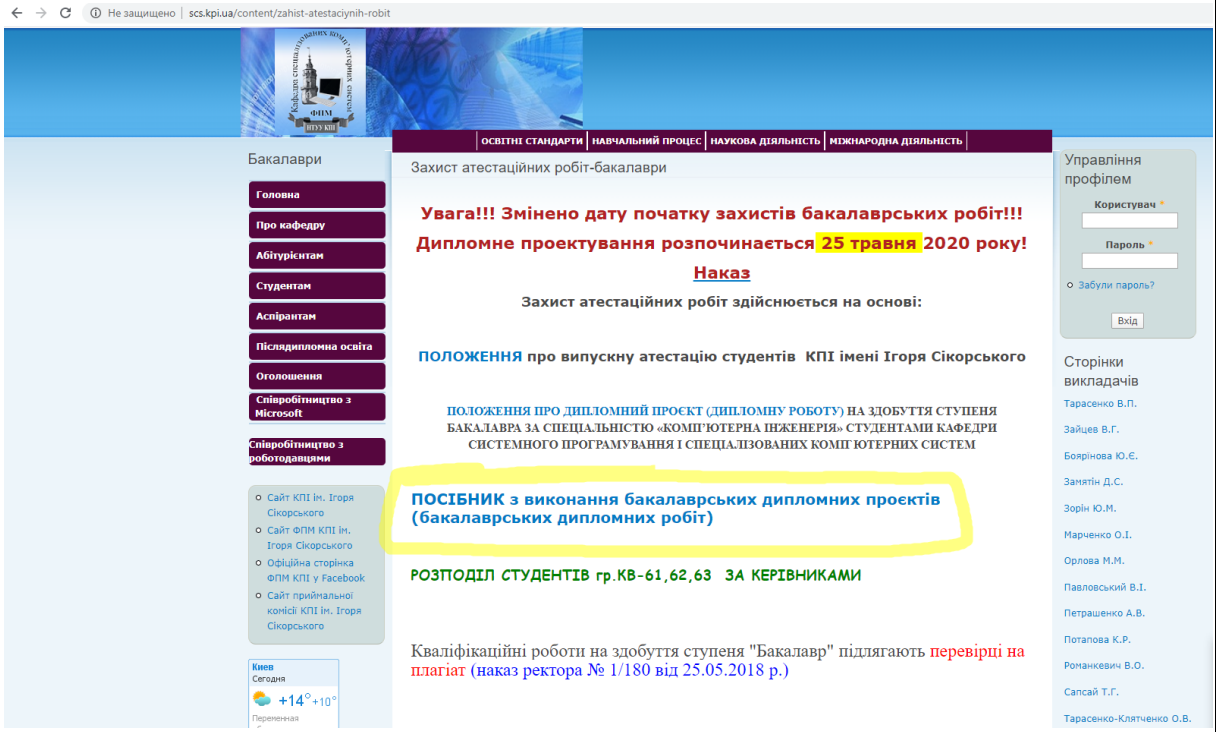


Рисунок 4.2 – Сторінка http://www.scs.kpi.ua/content/zahist-atestaciynih-robit, де знаходиться інформація для бакалаврів

Наш програмний код буде мати вигляд:

```
const qawolf = require("qawolf");

let browser, page;

beforeAll(async () => {
  browser = await qawolf.launch();
  const myContext = await browser.newContext();
  await qawolf.register(myContext);
  page = await myContext.newPage();
});

afterAll(async () => {
  await qawolf.stopVideos();
  await browser.close();
});

test("bakalavrInfo", async () => {
  await page.goto("http://scs.fpm.kpi.ua/");
  await page.click('ul:nth-of-type(2) [href="/"]');
  await page.click('[href="http://www.scs.kpi.ua/content/zahist-atestaciynih-robit"]');
  await expect(page).toHaveText('ПОСІБНИК з виконання бакалаврських дипломних проєктів');
  await qawolf.create();
});
```

Imports

Для тестування спочатку потрібна бібліотека qawolf, яка побудована поверх бібліотеки Playwright від Microsoft [13]:

```
const qawolf = require('qawolf');
```

beforeAll

Щоб розпочати тест, у блоці Jest beforeAll відбувається кілька речей. Тест запускає браузер і створює новий контекст, який представляє собою сеанс браузера у режимі "інкогніто". Цей контекст передається методу qawolf.register, щоб QA Wolf мав доступ до нього. Нарешті, створюється нова сторінка:

```
let browser, page;

beforeAll(async () => {
  browser = await qawolf.launch();
  const myContext = await browser.newContext();
  await qawolf.register(myContext);
  page = await myContext.newPage();
});
```

afterAll

Після закінчення тестування браузер закривається в блоці Jest afterAll. Якщо тест закінчився, запис припиняється, а відео зберігається.

```
afterAll(async () => {
  await qawolf._stopVideos();
  await browser._close();
```

});

Сам тест міститься у тестовому блоці Jest із вказаним ім’ям ("bakalavrInfo" у нашому прикладі). Тест спочатку переходить до вказаної URL-адреси. Потім він робить два кліки для переходу на сторінку http://www.scs.kpi.ua/content/zahist-atestaciynih-robit. В кінці перевіряється що на сторінці присутрній текст “ПОСІБНИК з виконання бакалаврських дипломних проєктів”

4.2 Підключення тестів до системи безперервної інтеграції

Для цього нам потрібно створити файл робочого процесу. Щоб створити файл робочого процесу для свого постачальника послуг CI, запустіть таку команду у своєму каталозі проектів:

npm init qawolf

Вам буде запропоновано вибрати свого постачальника IC із списку. Обираємо GitHub Actions.

Файл робочого процесу буде створений за адресою ./.github/workflows/qawolf.yml.

Тепер наші тести працюватимуть щоразу, коли хтось зробить комміт в будь-яку гілку нашого проекту. Файл /qawolf.yml буде містити:

```
name: myTest
on:
  push:
    # test every branch
```

```
    branches: "*"
  # schedule:
  #   - cron: "0 * * * *" # every hour
jobs:
  test:
    runs-on: ubuntu-18.04

    steps:
      - uses: actions/checkout@v2
      - uses: actions/setup-node@v1
      - uses: microsoft/playwright-github-action@v2
      - uses: actions/cache@v2

        with:
          path: ~/.npm
          key: ${{ runner.os }}-node-${{ hashFiles('root/package-lock.json')
}}
          restore-keys: |
            ${{ runner.os }}-node-

      -run: npm install

      # name: to start our local server
      #   run: npm run start & npx wait-on http://localhost:3000

      - run: npx qawolf test (--headless)
        env:
          FFMPEG_PATH: /user/bin/ffmpeg # to record our video
          QAW_ARTIFACT_PATH: ${{ github.workspace }}/artifacts
```

```
# LOGIN_PASSWORD: ${{ secrets.PASSWORD }}

- name: Uploading our Artifacts
  if: always()
  uses: _actions/upload-artifact@master
  with:
    name: qawolf
    path: ${{ github.workspace }}/artifacts
```

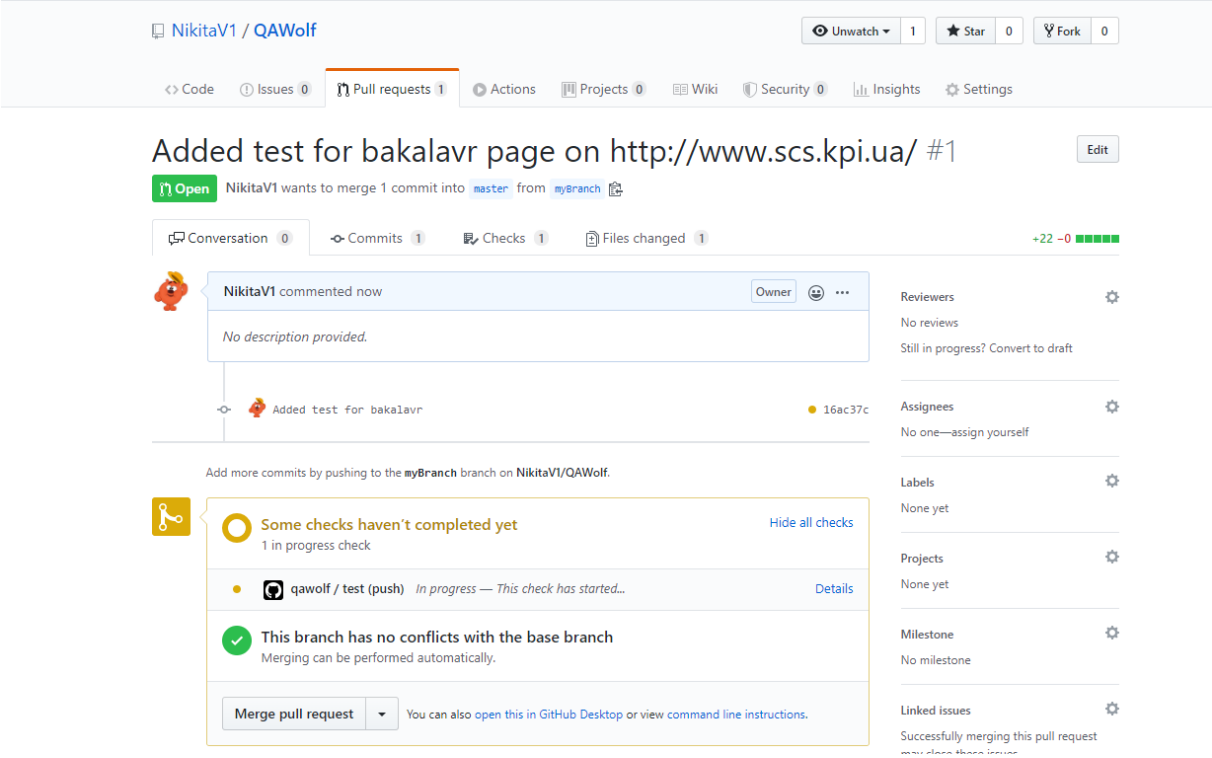


Рисунок 4.3 – Запуск автоматичний тестів після коміту до репозиторію

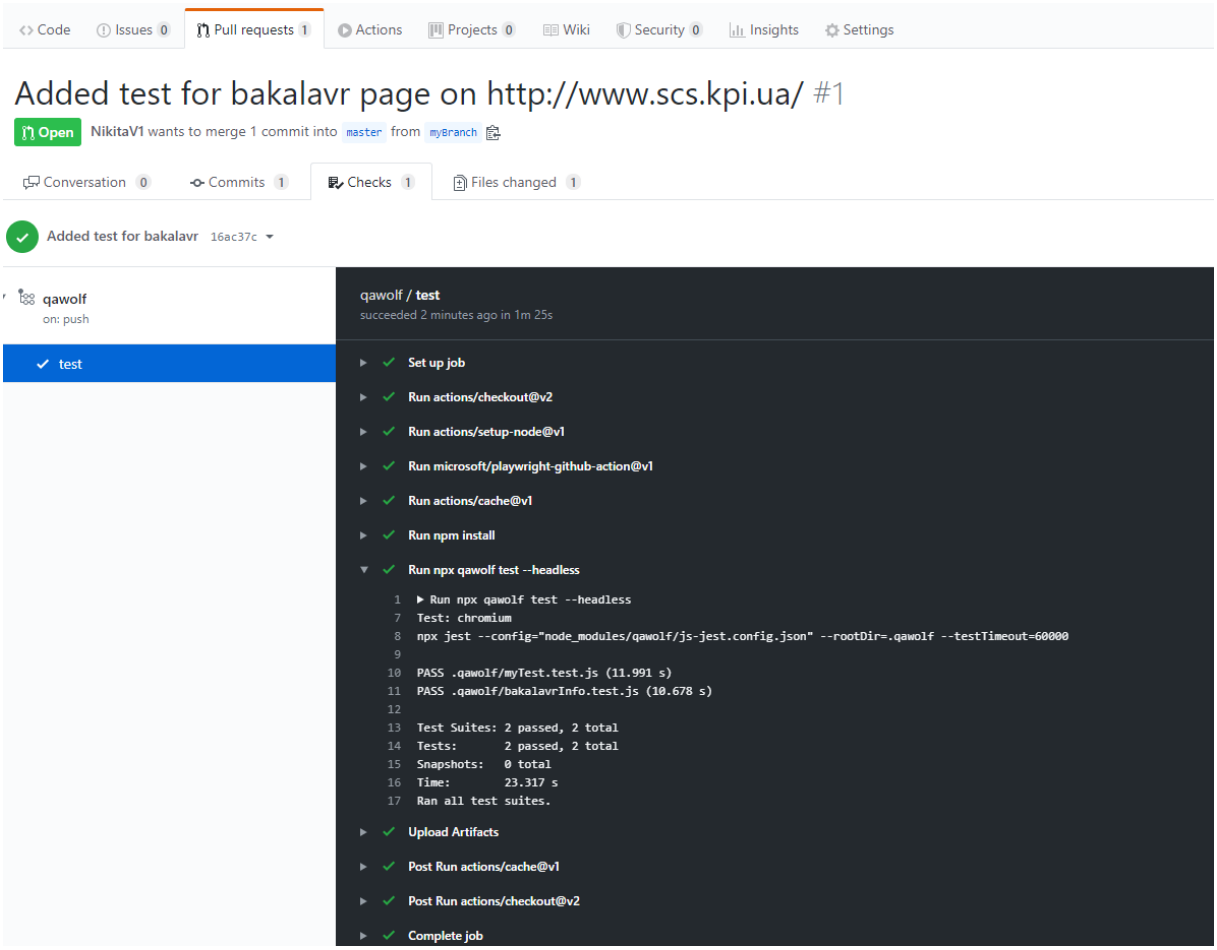


Рисунок 4.4 – Результат прогону тестів на віддаленому сервері

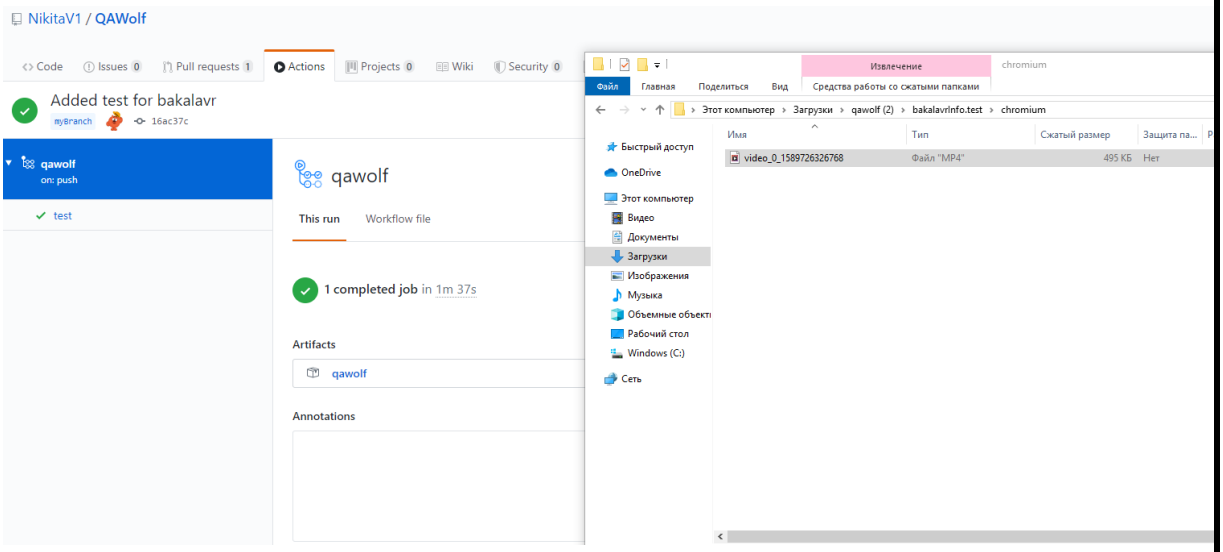


Рисунок 4.5 – Час виконання тестів і відеозапис результату

4.3 Програмна реалізація автоматизованої звітності до проєкту

Для звітності будемо використовувати месенджер Slack. Він безкоштовний і має інтеграцію з великою кількістю зовнішніх сервісів. Використовуючи його API, ми зможемо налаштувати систему звітності під додатки Android або IOS чи стаціонарну версію.

Ми створимо Slack webhook, що є URL-адресою, яка дозволяє нам програмно надсилати Slack-повідомлення. Зробимо запит POST до цієї URL-адреси, коли наші тести впадуть.

Спочатку нам потрібно створити програму Slack, яка відповідатиме за надсилання наших повідомлень про помилку. Почнемо з відвідування веб-сайту Slack API. У верхньому правому куті розташована зелена кнопка "Створити нову програму".

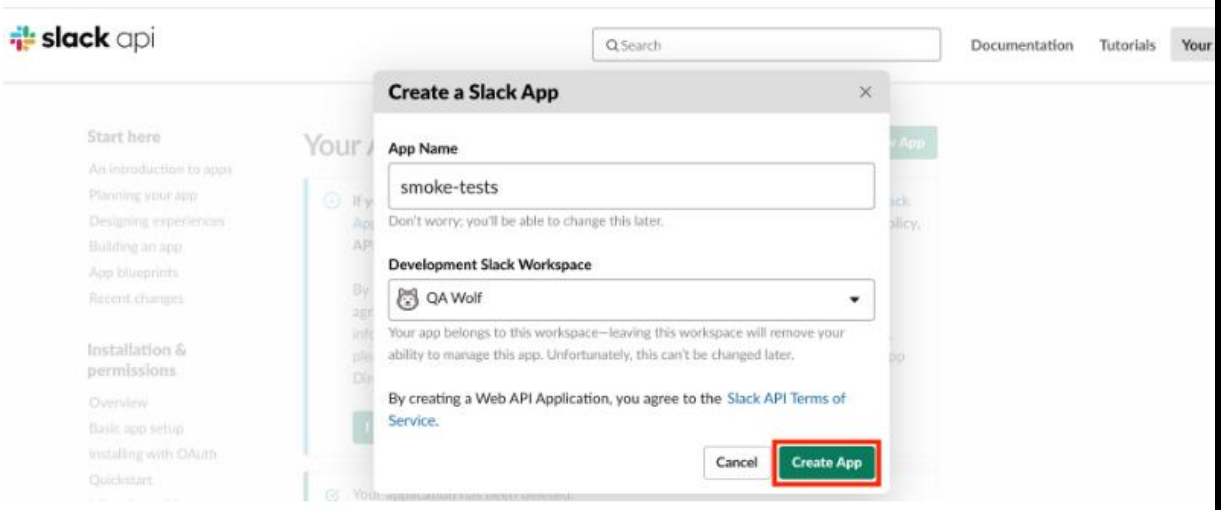


Рисунок 4.6 – Створення нової програми у Slack API

Далі потрібно зробити запит:

curl -X POST -H та 'Content-type: application/json' --data '{"text":"Your test, failed!"}' https://hooks.slack.com/services/SECRET

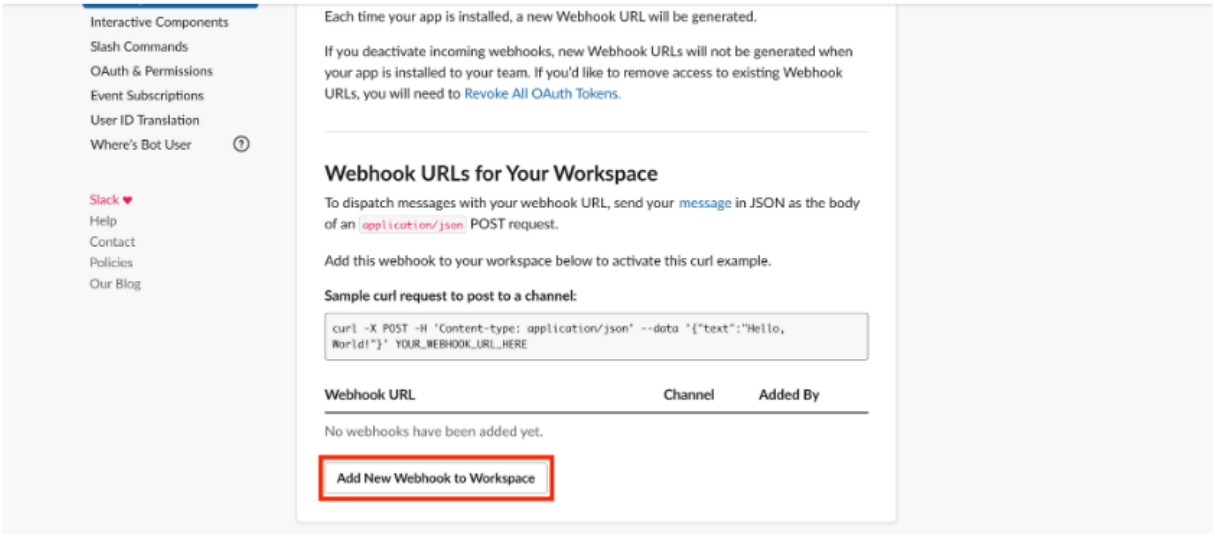


Рисунок 4.7 – Налаштування веб-кроку у Slack

Тепер, коли у нас є веб-крок Slack, нам потрібно оновити файл робочого процесу GitHub Actions. Ми додаємо крок, який робить запит POST, коли тести завершаються.

Замість того, щоб безпосередньо вставити URL-адресу у файл робочого процесу, ми додамо його до наших секретів сховища. Секрети - це зашифровані змінні середовища, які зберігають конфіденційну інформацію. Зберігання нашої URL-адреси веб-камери в таємниці заважає іншим бачити її та потенційно використовувати.

Додамо новий секрет у налаштуваннях вашого сховища. Встановимо значення SLACK_WEBHOOK_URL для URL-адреси Slack webhook.

Тепер оновимо наш файл робочого процесу. У нижній частині файла ".github / workflows / qawolf.yml" додамо наступні рядки. Ці рядки повідомляють GitHub зробити запит POST на Slack webhook, коли наші тести зупиняться через помилку.

name: Process of posting Slack Message to our account

if: failure()

run:

curl -X POST -H та 'Content-type: application/json' --data '{"text":"KPI main page tests failed!"}' ${{ secrets.SLACK_WEBHOOK_URL }}

Висновки до розділу 4:

Отже, в даному розділі був написаний тестовий сценарій для сторінки http://scs.fpm.kpi.ua/ , який було програмно реалізовано. Написано програмний код з використанням фреймворку QAWolf та підключення до системи безперервної інтеграції GitHub Actions. Здійснено реалізацію системи тестування на реальному вебдодатку кафедри. Отриманий результат підкреслив швидкодію роботи та написання автоматизованих тестів на мові JavaScript та зручність реалізації безперервної інтеграції. Було інтегровано месенджер Slack, на який автоматизовано приходить повідомлення про помилку під час виконання тестів.

ВИСНОВКИ

В ході виконання бакалаврської роботи були вивчені існуючі на сьогодні методи автоматизації тестування вебдодатків мовою Javascript, і на підставі цих даних була розроблена тестова програма.

Проаналізовані різні підходи до автоматизованого тестування користувальницького інтерфейсу, функціонального тестування, а також модульного тестування.

Основні висновки від з досліджень:

1. В часи швидкого розвитку інтернет технологій слід приділяти достатню кількість часу тестуванню надійності. Були розглянуті основні методи тестування програмного забезпечення, для задоволення цієї необхідності.
2. Було розглянуто основні види вебдодатків, їх призначення ті класифікацію. Виявлені вразливі місця вебдодатків відносно класифікації їх побудови. Також інструменти та технології їх створення.
3. Було розглянуто найпопулярніші на сьогоднішній день інструменти тестування вебдодатків відносно кожного методу, досліджено їх характеристики та швидкодію. Аналізуючи зібрані дані, було виявлено слабкі і сильні сторони методів та підрахована їх ефективність.
4. Було налаштовано систему безперервної інтеграції, яку можна застосовувати під час створення цілих проектів тестування.
5. Була обрана конкретна бібліотека і сервіс, яка необхідна для автоматизованого тестування прикладного програмного інтерфейсу вебдодатків. Здійснено реалізацію системи на прикладі сайту кафедри.

7 СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Вебдодаток і його характеристики [Електронний ресурс] https://www.centum-d.com/uk/veb-dodatok-yogo-harakteristiki

2. Introduction to DOM elements [Електронний ресурс] https://frontender.info/an-introduction-to-dom-events/

3. Сучасний веброзробник [Електронний ресурс] https://medium.com/nuances-of-programming/%D1%81%D0%BE%D0%B2%D1%80%D0%B5%D0%BC%D0%B5%D0%BD%D0%BD%D1%8B%D0%B9backend%D1%80%D0%B0%D0%B7%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D1%87%D0%B8%D0%BA-2018-a43d51a7bcd1

4. Фронтенд і Бекенд розробка [Електронний ресурс] https://skillbox.ru/media/code/frontend_i_backend_razrabotka/

5. 6 Different Types of Web Application Development [Електронний ресурс] https://www.clustox.com/6-different-types-of-web-application-development

6. Automation Testing Life Cycle [Електронний ресурс] https://www.lambdatest.com/blog/all-you-have-to-know-about-automation-testing-life-cycle

7. Компонентне тестування [Електронний ресурс] http://www.protesting.ru/testing/levels/component.html

8. Інтеграціонне тестування [Електронний ресурс] http://www.protesting.ru/testing/levels/integration.html

9. End-to-End тестування [Електронний ресурс] https://habr.com/ru/post/417395

10. Ranking Programming Languages by GitHub Users [Електронний ресурс] https://www.benfrederickson.com/ranking-programming-languages-by-github-users/

11. UI автоматизація, чому слід подивитись в бік JavaScript [Електронний ресурс] https://dou.ua/lenta/articles/automation-js-frameworks

12. JavaScript unit testing frameworks in 2020: A comparison [Електронний ресурс] https://raygun.com/blog/javascript-unit-testing-frameworks/

13. 11 Best JavaScript Unit Testing Framework and Tools [Електронний ресурс] https://geekflare.com/javascript-unit-testing/

14. QA Wolf [Електронний ресурс] https://docs.qawolf.com/docs/

15. Slack API [Електронний ресурс] https://api.slack.com/web